Let’s Not Learn by Trial and Error

Tracy Stauder
Managing Editor

Education has always been high on my list of personal and career goals. Over the years, my parents, who are retired schoolteachers, have helped me see the fun in learning and the importance of learning to be the best you can be. Though family commitments make it difficult to go to school in addition to work, I often find training opportunities in the workplace to fit my schedule and career goals.

But it is not just the love of learning that motivates me. I know that if I do not continually engage in training, my technical skill set will quickly become obsolete. The fast pace of progress in software technology leaves me no other option; I have to educate myself continually in the latest techniques and products.

Organizations generally recognize that they cannot afford to rest on their employees’ laurels any more than I can—successful strategies today may not be useful tomorrow. For this reason, the implementation of a solid training program is imperative to continued success.

But training is not something to be left to chance. In the words of Watts Humphrey, “Training must be planned, funded, scheduled, and required” (*Managing the Software Process*, Addison-Wesley, 1989). The core of most workplace training programs comprise formal educational environments—workshops, classes, seminars. Regular employee involvement in these opportunities yields significant returns on investment. Any manager who believes that cutting the cost of training is a good way to save money will find projects going over budget when early mistakes have to be corrected in late stages of production. As Paula Shafer points out, “the cost of good training is lower than the cost of not training or of training poorly” (page 6). Time and money is wasted when we learn by trial and error.

Formal, organization-level education should also be supplemented by individual study. Most work environments today provide access to the World Wide Web—a new dimension in educational and distance learning opportunities. I am continually impressed by the power of the Web and the library it puts at our fingertips. A fine example is the Software Technology Support Center’s Web site (http://www.stsc.hill.af.mil). Here you will find many software engineering educational aids and additional links to other defense software engineering sites (http://www.stsc.hill.af.mil/websoft.html).

Employees need to inform their management of their training needs. When possible, the employer should furnish employees with whatever they need to sharpen their technical tool set. Whether you take a TOPGUN approach to training (page 3) or you keep up with the latest trends via the Internet, we at CROSSTALK wish you success in improving your skills and in meeting your educational goals. ◆

CROSSTALK Needs Your Feedback, Letters

We at CROSSTALK thank all those who participated in our December reader survey. We appreciate the comments and suggestions offered; we are especially grateful for the kudos. The statistics provided by this survey will go a long way toward helping us determine what our readership needs and wants.

However, the final tally has not yet been calculated (surveys still arrive every day), so there is still ample time to complete the survey if you have not already done so. If you did not receive the December 1997 issue, or if the copy you have is missing the survey, you will find it on-line at http://www.stsc.hill.af.mil/XTalkSurvey.asp.

One of the preliminary results of our survey shows that readers enjoy the Letters to the Editor column. If you have read an article in CROSSTALK and feel that you have something to add to an author’s words (or if you disagree with them altogether) please take the time to compose a brief letter and send it to the senior editor, Sandi Gaskin, at gaskins@software.hill.af.mil. Though brief, these succinct opinion pieces clarify issues and raise important questions. Please don’t hesitate to make your voice heard.

The TOPGUN Approach to Software Acquisition Education

Mark E. Nissen
Naval Postgraduate School

The Naval Postgraduate School's TOPGUN approach augments the education of military officers with the leading principles, practices, tools, and techniques associated with software program management. The four-unit course involves student education, training, and research, in addition to direct experience through the TOPGUN software acquisition project. Because so many software acquisitions fail, this education and experience can be critical to surviving a program manager's first encounter with acquisition. As in the dogfight, surviving one's first software acquisition represents the key challenge to returning to learn from a second and third.

Software is not only ubiquitous, it is imperative. Information technology is deeply embedded within nearly every high-performance military system in use today, and many of the highest-performing weapons systems could not be mission effective without software. This applies to flight control systems, radar tracking and fire control systems, command, control, communications, computers, and intelligence systems, autonomous mission planning systems and many others. Even the infantryman is beginning to rely on software for enhanced battlefield performance. Indeed, the software share of weapons systems life-cycle cost has grown to be quite large, and it is growing at an increasing rate.

However, when compared with weapons systems hardware, software engineering represents a nascent discipline that reflects a high variance of methods, tools, technologies, and results. Coupled with steadily increasing computing power, software requirements perennially stretch the technological envelope and frequently are tied to schedules that challenge even the most mature engineering disciplines. Hence, software engineering represents one of the highest risk areas in modern weapons systems development, and its share of program risk increases exponentially with the growing dependence on weapons systems software. This makes embedded weapons systems software an extremely difficult acquisition problem.

The Naval Postgraduate School (NPS) has recently redesigned a graduate course from the acquisition curriculum to address this problem by augmenting the education of military officers with the leading principles, practices, tools, and techniques associated with software program management. The specific missions of this course are to help mitigate the difficulties inherent in software-intensive systems acquisition, to educate students in the principles and techniques of software acquisition and engineering, and to spark increased innovation in the acquisition process. The course, entitled "Embedded Software Acquisition," is offered to NPS graduate students—predominately military, but international officers and civilian professionals also attend—who are in residence at the Monterey, Calif. campus and enrolled in the program management (816) curriculum. Several elements of the course may also lend themselves to (synchronous) distance education.

The software acquisition course is comprised of four primary elements: education, training, research, and experience. Educationally, the course addresses the key principles of software acquisition and engineering, and takes a process-innovation approach. Students learn how and why software is unique and study textbook practice and practical cases associated with its engineering and innovation in the commercial sector and the Department of Defense (DoD). Training involves direct exposure to the DoD processes for software acquisition and assimilation of the best commercial practices and military lessons learned for procurement and program management. Students learn to critique—and perform in—the current acquisition system, with an emphasis on anticipating continuous acquisition reform and effecting process innovation. Through research papers, the students investigate the innovation aspects of software acquisition they are likely to encounter as program managers.

The experiential component involves "hands-on" usage of current tools and methods employed in software acquisition as students are given direct responsibility for managing and executing software development projects. These tools and methods include function and feature points, COnstructive COst MOdel estimating (COCOMO), Statistical Modeling and Estimation of Reliability Functions for Software reliability prediction, software capability assessment, metrics, and collaboration. Students learn to integrate and apply their relevant education and training through integrated product teams (IPTs) that are responsible for software planning, development, maintenance, and support.

TOPGUN Approach

The TOPGUN approach refers to the aerial combat tactics course that inspired a movie by the same name. Military research that focused on the disappointing aerial combat kill ratios of previous conflicts found that aviators who had experienced and survived even a small number of dogfighting engagements were exponentially more likely to survive their next one than were inexperienced pilots. Thus, the TOPGUN program was developed to provide a realistic training environment for students to *experience* and learn from aerial combat in a relatively safe environment.

Clearly, this approach is not limited to aerial combat tactics. The integration of principled education and realistic experience represents a well-accepted and effective pedagogical technique. This approach represents a centerpiece of the NPS software acquisition course, which as noted above, incorporates principles of software program management with the realistic experience of managing a software development project.

Because every graduate from the program management curriculum is increasingly likely to be responsible for some aspect of weapons systems software acquisition—even if assigned responsibility for a "hardware" program—this course helps to prepare them for their first encounter. In TOPGUN tradition, the students "train like they fight, and fight like they train." Moreover, because so many software acquisitions fail, this education and training can be critical to surviving the first encounter; as with the dogfight, we feel that surviving one's first software acquisition represents the key challenge to returning to learn from a second and third.

Course Requirements

Course requirements include readings, case analyses, an examination, a research paper, student synopses, and class participation in addition to the TOPGUN project. Each of the requirements for this four-unit course are summarized here; a more thorough description can be accessed through the course Web pages (http://web.nps.navy.mil/~menissen/mn3309).

Readings

Students are assigned more than a thousand pages of reading material that is required for adequate class preparation. They quickly learn that their (graded) class participation suffers if they fail to keep pace with the readings, and they are taught advanced volume-reading techniques for assistance with this requirement. Readings include the Software Technology Support Center *Guidelines for Successful Acquisition and Management of Software-Intensive Systems* as a text, numerous General Accounting Office cases that address software management, an abundance of on-line readings, and handouts from current journals and periodicals. Although the course is primarily directed toward education as opposed to training, the assigned material also includes the Defense Acquisition Workforce Improvement Act content from the software acquisition management series.

Case Analyses

As noted above, numerous software management cases are assigned as an integral part of the course readings and integrated into case-method lectures. One or two encompassing cases are thoroughly analyzed and presented by students working in small teams (three or four people). The purpose of the case analyses is to ground students' course knowledge in complex and relevant software programs. This requires the application of concepts, principles, tools, and techniques learned in class in addition to the analysis of real-world programs and design of program-management solutions.

Examination

A comprehensive examination tests students' ability to integrate the readings with lectures, cases, and discussions and to critically apply the principles, concepts, and applications covered in the course. The examination mostly tests students' higher-level knowledge and abilities (as opposed to narrow details) and requires understanding, application, synthesis, and some creativity and design (as opposed to short-term memorization and "data dumping"). Unlike case analyses and the course project, which involve teamwork, the examination measures only individual effort.

Research Paper

Each student must write a research paper (approximately 5,000 words) that investigates a relevant aspect of software acquisition or engineering innovation. This helps students specifically tailor their learning to a topic of direct or immediate relevance. It also provides the opportunity to work toward innovation, the results of which can be taken directly to the job and applied after graduation. Recent papers have investigated the use of requisite variety for weapons systems prioritization, agent-based contract management, reengineering the software acquisition process, and other personalized investigations. Because the course is taken midway through the master's program at NPS, students are encouraged to focus on topics that help them orient and focus their thesis research.

Student Synopses and Participation

Oral and written communications represent important managerial skills for military and business leaders, and with the advent of oral contracting, critical listening and interaction skills are now becoming increasingly important in acquisition. To practice and enhance these skills, students work in two-person teams to prepare written synopses of selected course material, then present this material to the class. Presenters are expected to effectively communicate the key elements of this material to classmates, who must in turn interact with the presenters to demonstrate a reasonable understanding and appreciation for the key points. Active class participation is strongly encouraged, and students strive to relate their relevant operational experience to the principles and practices covered in class.

TOPGUN Project

In addition to accredited graduate education and the requirements outlined above, the TOPGUN project probably best distinguishes this software acquisition course from others. The project entails team activities that involve software acquisition and software engineering. Half the teams serve as program management offices (PMOs) for Web-based software projects and the other teams perform as contractors to develop the required software systems. Although the software is technically simple and the project scope is small—as is appropriate for a quarter-long course—the students must go through most of the required acquisition and engineering steps, e.g., a mock Defense Acquisition Board and Milestone II review, Software Development Plan, Preliminary Design Review and Critical Design Review, Request for Proposal preparation and analysis, proposal preparation and source selection, in addition to managing the development of the software. Many outside software professionals have commented on the realism of the project environment, particularly in terms of the ambiguity, uncertainty, schedule pressure, metrics-tracking, and IPT-coordination difficulties that are encountered even on this technically simple project.

As an interesting twist (and pedagogical departure from reality?), the incrementally developed software is fielded at the project midpoint—midway through the course, ready or not! The teams then switch roles and maintain the software in a post-deployment software support (PDSS) mode; that is, those on the PMO teams become "contractors" and maintain the software, while the original contractor teams take over the PDSS activities, such as site operation, training, and enhancement. The students never fail to express their amazement of how this change of perspective affects their attitudes, and students on both sides of the contract quickly reflect back on their performance in the other role with greater empathy and understanding of how interrelated software acquisition and engineering truly are.

The project involves developing a Web site on software acquisition (http://web.nps.navy.mil/~com) that is not intended to be spectacular in any respect; rather, it provides a focus and some minimal technical challenge for the course project, and it requires the students to integrate, organize, and present their learning about software management to the world. Students take this requirement seriously—although they appear to enjoy the project tremendously—and produce insightful summaries of key principles, practices, tools, and techniques associated with software management. They also provide a valuable set of lessons learned for future classes as an approach to organizational learning.

Results

This course produces a number of results. At its conclusion, many students express feelings of confidence combined with humility, for example. They seem to appreciate the power and value of advance planning, close integration of software acquisition with engineering, teaming, metrics, and "inch pebble" (small milestone) management, and they feel much better prepared to take on a software management job after graduation. At the same time, they appreciate how difficult it can be to manage a technology-focused IPT and program, even for a technically simple product. Faculty colleagues comment on the students' comfortable familiarity with software concepts in their other courses and their ability to extend some of the useful techniques to non-software program management problems.

In the end, nearly all students indicate that they are better prepared to manage software-intensive programs as a result of the course—and I concur. If they can "come back alive" from their first engagement with a *real* software program, all the work required to make this kind of course effective will pay off. With students from the first class graduating this year and being assigned to challenging weapons systems programs, we should soon see whether our TOPGUN approach to software acquisition fulfills our high expectations for it. As a professor dedicated to constant improvement, I welcome any feedback that can help further this goal. ◆

About the Author

Mark Nissen is a professor of information systems and acquisition management at the Naval Postgraduate School and leads the school's program of acquisition research. He is the course coordinator for MN3309, Embedded Software Acquisition, and also teaches courses on decision support and information technology acquisition, in addition to advanced graduate seminars on process innovation, intelligent agents, and like topics. His research is directed toward the application of knowledge systems to innovate acquisition processes, with current work focused on intelligent acquisition agents. Before beginning his academic career, he acquired a dozen years management experience in the aerospace industry and was a supply officer in the Naval Reserve.

Naval Postgraduate School
555 Dyer Road, Code SM/NI
Monterey, CA 93943-5000
Voice: 408-656-3570
Fax: 408-656-3407
E-mail: Mnissen@nps.navy.mil

Planning an Effective Training Program

Paula S. Shafer
Independent Consultant

In Quality Is Free, *Bill Corsby notes that the cost to build a product correctly (cost of conformance) is lower than the cost to fix the product after delivery to clients (cost of nonconformance). The same concept is true in training: the cost of good training is lower than the cost of not training or of training poorly. Organizations stand to lose significant amounts of money from lost productivity when there are changes in process, technology, or culture and employees are not properly trained to handle them. Some organizations report a 3,000 percent to 6,000 percent return on investment from good training [1]. By contrast, poor training or poorly planned training wastes money and time and lowers morale. A properly planned training program is required to ensure success and return on investment for training dollars.*

The first thing to ask when planning a training program is "What do people need to do their jobs?" The answers to this question usually fall into the following categories:

- Process – a way to work.
- Technology – the tools with which to work.
- Management support – a reason for the work.
- Skills – ability to do the work.

Process addresses what work to do, how to perform the task, when to do it, what resources are required, who has the inputs, and who gets the results. Without process, there is chaos. Training alone cannot establish process, but process improvement is at best transitory without good training. Also, ineffective, inconsistent, or undocumented processes require more training to overcome the confusion or miscommunication that is rife in immature organizations. Employees who attend training and return to an undisciplined environment will often not use the skills learned. The money spent on the training will have been wasted and morale will suffer.

Technology. People need appropriate tools and technology to perform their jobs. An organization and its management select, purchase, and make available the appropriate technology. The company's process indicates appropriate uses for the technology. Training tells the employee how to successfully use new technology.

Management support is needed to provide the motivation for effective organizational change. An effective reward system reinforces desired behavior and corrects undesired behavior. Although some organizations try to *motivate* through training, there is little or no lasting impact with this approach. Courses that attempt to motivate have objectives that use phrases such as "understand" or "provide an overview" or "gain an appreciation." Participants may leave this kind of training enthused, but a week later they are back to old work patterns. Training cannot be effective without management support consistent with the messages in the training program.

Skills and knowledge are where training can have the most impact in an organization. Skills that people need overlap with process and technology and are reinforced through effective management support. Training is not the solution to problems that businesses have today; however, without training, an organization will fail in its process improvement program.

This article defines effective training, discusses various means to deliver training, and suggests possible metrics to evaluate training effectiveness.

Training Goals

Effective training is integrated and consistent with many aspects of a software development group. It must be consistent with the following:

- *Organizational goals and strategy.* Although training does not define or establish business strategy, it is important to reinforce that strategy at every opportunity, including during training. Training developed and delivered within the organization should always begin with the business goals. Any training purchased from outside can be aligned with those goals if it is introduced by senior management, who reinforce the mission and vision.
- *Project planning.* The Organization Standard Software Process (OSSP) software project planning procedures need to address project training needs. For example, software development planning standards should account for training to encourage the planners to consider what skills and abilities their project team members will require.
- *Software quality assurance (SQA).* An independent SQA program must be highly involved in training. Independent SQA can perform multiple roles for a software development organization: verifying the implementation of processes, mentoring projects in the use of processes, and collecting and analyzing data on the quality of product and process. Integrating training and SQA means several things. First, some training may be delivered by the SQA organization. Second, SQA participates in the review of training materials to ensure that the messages of the training are consistent with the organization's standard software process. Third, SQA ensures that

project teams receive the training that had been planned. Finally, as SQA analyzes process and product quality, it may propose additional training or improvement of existing training.
- *Software process improvement (SPI) initiatives.* The organization's SPI program should address training in multiple places. The Software Process Engineering Group (SEPG) ensures that process training is integrated into the OSSP. It defines a process to plan, to make available, to track, and to measure quality of training. The SEPG also coordinates or delivers training on topics related to process. Examples of training that the SEPG addresses are the Capability Maturity Model, process analysis, process modeling, or using specific processes during pilots and implementation.

Theory vs. Practice

Effective training focuses on skills or competence in the software development organization. It is specific, timely, and result oriented. People learn by doing, but many trainers do not seem to realize this. Many training techniques do not include active involvement by participants; instead, participants spend most of their time listening to an expert lecture on the theory of topics like SQA or project planning. Precious little time is devoted to practicing these theories. One colleague of mine calls this the "spray-and-pray" approach: the lecturer disperses knowledge, and management hopes it will somehow be absorbed. What usually happens instead? If the participants can stay awake, they may learn something, but there is no way to test what they learned until they return to the workplace. When the real world collides with the theory, the theory will not be applied, no one is available to help the employees apply it, and old methods or habits are perpetuated. The training dollar is wasted, and the employees become discouraged.

In an effective training course, real-world experience and applying skills are more important than theory. The goal is *not* to put employees through training; the goal is for employees to learn and begin to apply new skills. Less time is spent in lecture and more time is spent practicing the skill. A good approach is the case study. For example, a course on peer reviews could provide participants with the experience of a peer review through sample materials for a simulated software inspection. A better approach is to use actual materials from the trainees' projects. For example, a former colleague of mine delivered analysis and design principles training to an entire project team using its own project rather than a case study. The training was spaced over a period of time as well: teach a little, then work a little. The skills could then be used in "real life," and the trainer could be questioned upon returning to the classroom after the trainees tried out the concepts. When the team completed this training, it had draft work products that were used as they progressed in the project.

Methods of Training

Once a careful analysis of the training needs has been accomplished, appropriate training can be developed. Several methods of delivering the training are viable in today's environment.
- Classroom.
- Teletraining.
- Videotape presentations.
- Job aids or just-in-time training.
- Mentoring.
- Computer-based training.

All have value when used appropriately. To gain the most out of training, use all the methods that make sense for the organization's needs.

Classroom training is the traditional delivery approach and may also be called a workshop, lecture, or laboratory. This approach relies on an instructor who leads a participant discussion and usually a case study or exercises. This mechanism is the most flexible and easiest to adapt. It can be developed comparatively quickly and inexpensively. Drawbacks are lost work time for the participants, scheduling difficulties, travel costs, and the lack of skilled instructors.

A close approximation to training in the classroom is teletraining or using teleconferencing facilities for a lecture or laboratory-type course. This is useful in the same ways as classroom-led training. Additionally, it can be delivered to isolated or dispersed locations while minimizing travel costs. This does, however, require a significant investment in technology, and it still requires that people leave their work site. Also, teletraining can present scheduling problems, especially when the participants are located in different time zones. Teletraining also is difficult to make interactive and limits the instructor's available techniques. It is best used for short, clear, and concise training.

Videotaped training is gaining popularity in some organizations. This has some of the same advantages of teletraining in that the training can be delivered to remote locations but does not require the significant investment in technology required by teletraining. It is excellent for delivering short, clear messages and can incorporate interviews or demonstrations from managers or staff, especially when used in conjunction with other forms of training. The disadvantages are the high development and production costs and the lack of participant interaction. Generally, this form has limited utility if used without on-site support.

Job aids, also called just-in-time training, are items such as cards or trifold brochures that outline a procedure succinctly. These are a useful means of training—relatively inexpensive to develop, and easily modified when needed. This form of training does not require that participants leave the job, thus minimizing costs such as lost productivity or travel. These aids can be used to supplement any other form of training, especially for topics like procedures and technology. They are not effective stand-alone, especially for complex skills like project planning. Additionally, unless carefully designed, they can be complex and difficult to comprehend.

Highly skilled people who guide the learner in the workplace perform mentoring. This approach is inexpensive to develop because the expert already has the skills. It is provided at the time the participant needs it and does not require leaving the workplace. It does require some investment, though. The mentor's instructional skills need to be developed, and a structure for the training, such as a "lesson plan," needs to be developed. Quality control is difficult because there is usually no evaluation of it. This form of training works only when properly planned to ensure that the mentor is available, able, and motivated. To merely give training participants the name and telephone number of the expert to call if they have problems is *not* mentoring. The danger in this approach is the potential to propagate bad practices if the mentors do not apply best practices.

Computer-based training is useful in some situations. If well designed, it can be interactive, be used in dispersed geography, minimize travel costs, and reach a wide audience quickly. It is especially useful in simulating dangerous or costly situations, such as landing a jet on an aircraft carrier. However, it requires significant development costs and lead time and can have significant delivery costs when simulating complex environments. It is good for learning basic skills but falls short when used to teach advanced skills. Eventually, the "pilot" has to try to "land a real airplane." This form of training is usually difficult to maintain, update, and redistribute.

All of these forms of training have strengths and weaknesses. When appropriately integrated, they can capitalize on the strengths of each while overcoming their individual weaknesses. The result will be the development of a coherent and comprehensive training capability.

Measuring the Quality of Training

There are at least four ways to measure the quality of a training class or program: post-course evaluations, testing, follow-up surveys, and the organization's metrics program.

Most training courses end with a post-course evaluation completed by the participants. These forms ask the participant's opinion of the course materials, the instructor, and the classroom environment. But it can be difficult to know precisely what these evaluations measure. Do they evaluate whether the participants learned anything or whether they merely enjoyed the training? For example, studies show that these evaluations show high marks when the instructor tells jokes. However, studies also show that when participants enjoy the experience, they open their minds to the new process or procedure, which is a key step to change behavior. Unfortunately, post-course evaluations do not measure the effectiveness of the training in changing behavior in the workplace.

Testing is an effective means to evaluate whether a participant learned. However, this form of evaluation requires investment in developing good tests. This form of evaluation is needed in certain circumstances, such as maintaining accreditation with an organization like the American Council on Education. The tests allow employees to achieve credit for the training in a university environment. However, many organizations find the value of the credit is not worth the expense.

Follow-up surveys are an improvement on the post-course evaluation. The approach is to send an evaluation to participants some time after the training was delivered, e.g., six to nine months later. These surveys are generally more in-depth than most end-of-course evaluations and are targeted to the goals of the training course. Such a survey would ask questions such as "Have you defined and are you using the procedure for making the software project size estimates?" These surveys only secondarily seek to determine the participant's opinion of the training. The focus is on the behavioral change in the work environment. These surveys also can gather data concerning other factors discussed earlier, such as process, technology, or management support. The chief drawback to follow-up surveys is the cost to develop and conduct the survey. It also is difficult to get people to respond to surveys; those who do respond may not represent the average participant.

The best way to judge a training program is in conjunction with the organization's metrics program. Collect appropriate data before and after conducting training. Effective training will result in improvement in the data. As an example, software inspection data reveals that 75 percent of defects found in design, code, and test phases of the software lifecycle relate to defects in requirements. Six months after implementing a training program in requirements elicitation and analysis, the defect rate goes down to 35 percent, and the training can be declared a success.

Summary

Training is necessary to improve process, but it costs money. Poor training potentially wastes money, lowers morale, and reduces productivity. Good training achieves significant returns on investment. To get the most out of your training investment,

- Deliver training aligned with business strategy.
- Focus on skills that people need.
- Emphasize interactive training.
- Provide multiple forms of training.
- Reinforce through management support. ◆

About the Author

Paula S. Shafer is a freelance consultant and trainer specializing in software project management and software process improvement. She has nearly 25 years software development experience. She is currently developing a course in requirements management.

220-K Stony Run Lane
Baltimore, MD 21210
Voice and Fax: 410-366-6430
E-mail: psshafer@erols.com

Reference

1. Wiggenhorn, William, "Motorola U: When Training Becomes an Education," *Harvard Business Review*, July-August 1990, p. 75.

Intelligent Agents for Internet-Based Military Training

Niraj Joshi and V.C. Ramesh
Illinois Institute of Technology

In a project sponsored by the U.S. Army Research Office, we developed and delivered an Internet-based multi-agent system for military training. We describe the client/server-based software architecture of this multiuser intelligent agent (logobot) application that focuses on bookmark automation. We pinpoint some software design issues, such as the inevitable trade-off between real-time response and the complexity of the knowledge base, which will likely be encountered in similar Internet-based military training applications.

The U.S. military has several training facilities geared toward military personnel who are geographically distributed. Like other military units, the U.S. Army offers mail-based correspondence courses from battle laboratories such as Combined Arms Support Command (CASCOM) in Fort Lee, Va. Although there is no core curriculum, students enroll in different courses based on their military occupational specialty (MOS). The project described in this article is indicative of the steps the Army is taking to migrate from paper-based to electronic training by way of the Internet.

This project provides instructors with tools to gather and categorize Internet uniform resource locators (URLs), and lets students choose categories of URLs that can be automatically stored in their bookmark (favorites) collection. The instructor can exercise control over training materials accessed by students and, as a secondary benefit, can restrict unnecessary "net surfing." Because one of the aims was to provide interdisciplinary training to students, the subject categories for organizing the URLs, which we labeled *domains*, cut across the students' MOS categorization. They can subscribe to any domain they wish, but can only access the URLs provided by the instructor. This effectively constitutes one use of an Intranet wherein the instructor employs the Internet as a source of training material to be housed in an Intranet server that students access.

We devised a software architecture based on the emerging technology of intelligent software agents that we call *logobots* (from *logos* [Greek for *knowledge*] and *bots* [Internet jargon for *intelligent agents*]). This multiagent system has been delivered and installed at CASCOM. This project demonstrates the capabilities added by agent technology to the more traditional client-server software architectures that are commonplace in military applications. We highlight some of the design issues that are likely to be faced by other software engineers who develop Internet-based training applications.

Logobots

An intelligent agent is a software program that autonomously senses the environment, acts upon it, and over time acquires competence by learning from the environment [1]. Depending on their functionality, agents can be described differently. An agent that acts as a personal assistant is called an *interface agent* in [2]. An agent that has access to at least one and potentially many information sources, and is able to collate and manipulate information obtained from these sources and respond to user queries, is called an *information agent* in [3].

Logobots, which are best described as task-specific autonomous software agents [4], are both interface agents and information agents, as we will elaborate below. One logobot is assigned to each instructor or student, and each augments its knowledge base as new information becomes available. Each logobot maintains training information for its user and has user-interface capabilities to acquire and maintain a knowledge base that pertains to each individual and training domain. We employ two types of logobots: *student logobot* and *instructor logobot*.

The student logobot acts as the student's personalized interface to the remote training. It uses an acquired knowledge base of preferences and reference material and maintains each student's bookmarks. Salient features enabled by this agent include

- *On-demand* or automated update of training material.
- An intuitive interface to the training material, which includes the ability to search the student's bookmarks by specifying domain, URL, title, or description.
- Personalization by selection of training domains and other preferences.
- Ability to change the default preferences assigned by the instructor.

These capabilities provide flexibility for students to identify their training needs. This not only motivates students but also helps them explore the paradigm of *learning to learn* and *just-in-time learning.*

The instructor logobot acts as both an interface agent and an information agent. It helps instructors easily acquire training information from various sources and automatically disseminate it appropriately to the students. It can search the Internet and helps instructors filter relevant information and classify it into different domains. It also assigns default training preferences for different classes of students. Multiple instructor logobots maintain a centralized knowledge base for various training domains. Each instructor logobot can be personalized. Salient features enabled by this agent are

- An *Internet search interface* based on Java client-server technology, which uses popular search engines like AltaVista to locate training information.
- Filtering of the search results and classification of filtered sites. Information about the state of each document or site is maintained as a part of the central knowledge base; the agent maintains and uses this information to filter out previously seen URLs.
- Facilities for management of domain information, student information to link students to the domains, and for setting the default preferences.

System Architecture

Logobots are a multiagent system that contains agents and a centralized repository [5]. It is designed to operate in a distributed multiuser (Internet) environment. From a knowledge engineering viewpoint, this architecture could be considered a blackboard architecture [6], which is implemented using client-server tools and techniques. Agents (logobots) act upon the central knowledge base according to user requests and make changes to the repository according to the state of repository data. Logobots have unique identity and built-in security and authentication information. Users need to authenticate before logobots let them access the central knowledge base.

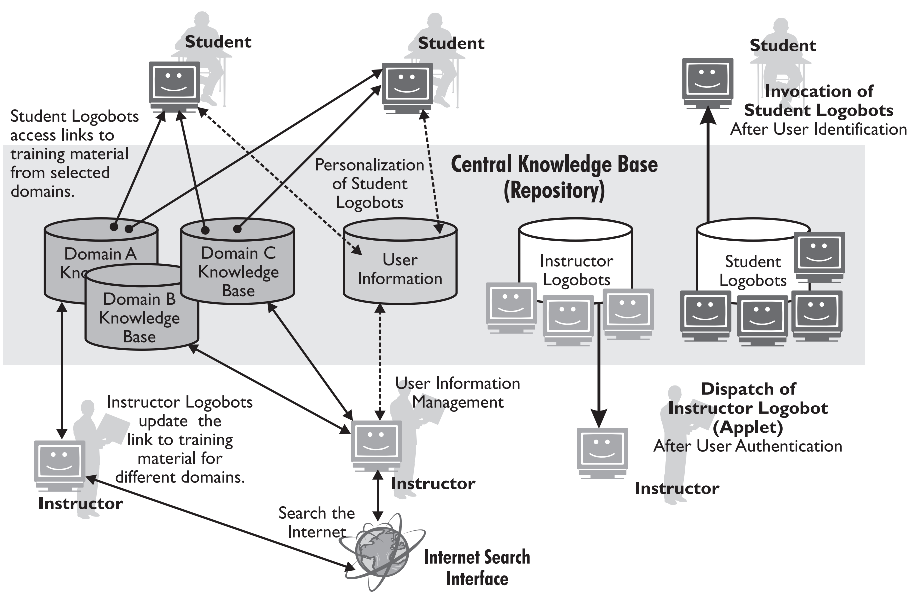
Figure 1 illustrates the system architecture. After proper authentication, students or instructors invoke their logobots. The central knowledge base, which is resident on a networked server with an operational Web server, contains domain information and training material specific to the domains, as well as user or logobot information; the current *state* of the knowledge base, which includes the *accept bin* and *reject bin,* is for memorizing processed information.

Figure 2 displays the internal architecture of the logobots and shows the various interactions between their modules and the central knowledge base. As previously indicated, all logobots are dispatched from a central site (Web server), which houses the central knowledge base and its access modules. Once the logobots are launched into the user's Web browser, they display a personalized initiating page for the user. The user can navigate through the system, accessing various training materials in different domains. Both of the logobots provide a keyword-based search facility over the domain knowledge base. Use of the *search and personalization* mechanism allows students to individually choose their preferences for the sections of training material and to search the material for references.

Instructor logobot supports additional functions to maintain the central knowledge base and manage the user information. Through the *Internet search interface*, the instructor can find training material over the Internet. The *crawler* component contacts various information sources on the Internet and gets the URLs. These URLs are filtered and classified by the *comparator* component, according to the state information in central knowledge base. Exact matches are eliminated, and if necessary, the results are shown to the instructor for further filtering. The instructor may add URLs to certain domains and reject others. These instructor activities are recorded and used later for more efficient search, filtering, and classification.

System knowledge is augmented every time the user performs an action, and the system learns from both the user and the environment. The student and the instructor logobots are client-side components implemented using Java applets, JavaScript, and plain HyperText Markup Language, whereas the central knowledge base is a server-side component implemented with common gateway interface (CGI) programs. The client-server type search modules are implemented using Java applets and applications.

Figure 1. *Overview of the multiagent logobots system.*


Student
Student
Student
Student Logobots access links to training material from selected domains.
Personalization of Student Logobots
Central Knowledge Base (Repository)
Invocation of Student Logobots
After User Identification
Domain A Knowledge Base
Domain B Knowledge Base
Domain C Knowledge Base
User Information
Instructor Logobots
Student Logobots
User Information Management
Instructor Logobots update the link to training material for different domains.
Dispatch of Instructor Logobot (Applet)
After User Authentication
Instructor
Instructor
Instructor
Search the Internet
Internet Search Interface

Design Issues

Instructor logobots perform filtering and classification functions based on a learning algorithm, which is a simplified version of a technique called memory-based reasoning (MBR) [3]. The search interface monitors the selection and rejection of bookmarks whenever the instructor is adding training material. Bookmarks and related information are stored in accept and reject bins and later used to filter, classify, and search for new material.

With a little modification, advanced learning techniques like neural nets or semantic nets could augment this simple learning scheme. However, this software had to be usable at CASCOM

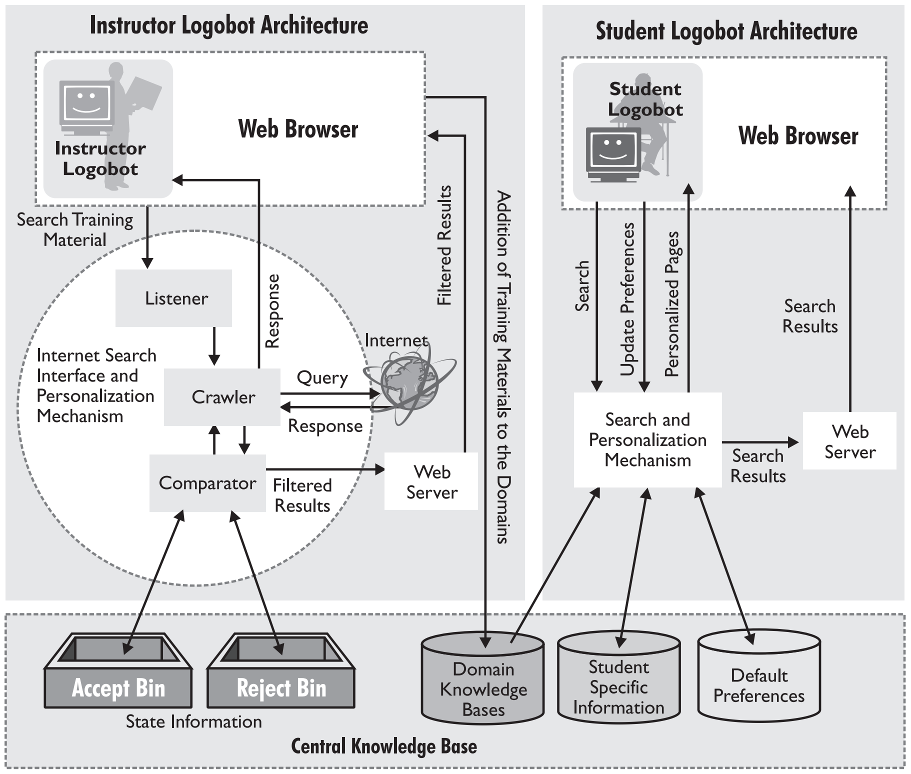


Figure 2. *Internal architecture of logobots.*

as quickly as possible after installation; upon considering the time it takes to train instructors in these advanced schemes, we implemented the simpler MBR scheme. Such trade-offs are often inevitable with intelligent agent-based software systems.

The decision to use the MBR learning algorithm led to another design trade-off: response time vs. the size of the state information stored in the accept and reject bins in the central knowledge base. Recall that for each URL, at a minimum, state information includes the title of the corresponding Web page. URLs in the accept bin also include the domain name. One also could add descriptive information about the contents of the Web page. Clearly, the number of URLs and the complexity of the information stored for each will determine the size of the accept and reject bins. The larger the size of these bins, the better the filtering of the URLs. However, larger accept and reject bins also lead to poorer response times and greater delays, which leads to potential user frustration.

How many URLs should be stored? It is difficult (if not impossible) to determine a generic solution to this design trade-off problem. Based on empirical tests with this implementation of the logobots system, we limited the size of the bins to 100 URLs each. We also limited screening to exact URL matches and excluded titles and descriptions from the filtering process. The trade-off between the size and complexity of knowledge and search response times is a problem that pervades the knowledge engineering field.

A relatively minor but nevertheless important challenge was the need to work across "firewalls." We wanted to minimize use of the inefficient CGI protocol and rely on socket communication primitives provided by Java. However, for Java applets to communicate with the applications running on the central knowledge base server, firewall administrators must provide trusted ports. This was not a problem in this particular military application, but security will play an important role in the design of any such agent system, especially for military applications.

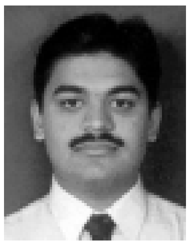
The last design issue concerns why we chose the Internet and Java. When we started creating this system, Java and the Web were unproven technologies. However, these technologies turned out to be advantageous for several reasons. One of the main benefits was the ease with which our Java-based instructor logobots could access popular search engines, such as AltaVista, and build on those results for URL filtering. This prevented the need to create a new search engine and let us concentrate on the higher-level filtering and user interface aspects. We believe that such Internet-based agent applications that leverage the client-server infrastructure of the Web will enable the military to migrate its distance learning facilities to the Internet.

Conclusion

Intelligent software agents are now migrating toward mainstream software systems. We have described one such system for increasing the efficiency of military training. The system can be particularly valuable in controlling the rapid influx of information in times of crisis (such as a war) by filtering and classifying voluminous event reports into pre-defined subject categories, with students receiving only the portion of information relevant to their line of duty. It also reduces the costs involved in providing military training when compared to the traditional paper or CD-ROM-based techniques.

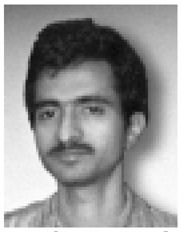
Though our main focus has been to automate the process of Internet-based military training, the resulting multi-agent framework could be used for any multiuser Internet- or Intranet-based application, which requires controlled dissemination of structured information. The system is not limited to URLs, but is readily extensible to other multimedia objects. Logobots are a step in the military's moves toward virtual classrooms. ◆

About the Authors



Niraj Joshi is a graduate student pursuing a master's degree in computer systems engineering at Illinois Institute of Technology. His research interests include intelligent agents, software engineering, and database systems. He has three years experience in design and development of software systems based on the Internet and databases. He has a bachelor of engineering degree in computer science from M.S. University in Baroda, India.

Illinois Institute of Technology
Electrical and Computer Engineering Dept.
3301 S. Dearborn Street
Chicago, IL 60616-3793
Voice: 312-567-5074
Fax: 312-567-8976
E-mail: npjoshi@ece.iit.edu
Internet: http://www.ece.iit.edu/~npjoshi



V.C. Ramesh is an assistant professor at the Illinois Institute of Technology. His research interests include intelligent agents, software architectures, and simulation. He has been the principal investigator for several research projects sponsored by various Department of Defense agencies, such as the Army Research Office and the Office of Naval Research. He has published more than 30 technical articles. In 1996, the National Science Foundation named him a recipient of the CAREER award. He holds a doctorate in electrical and computer engineering from Carnegie Mellon University.

Illinois Institute of Technology
Electrical and Computer Engineering Dept.
3301 S. Dearborn Street
Chicago, IL 60616-3793
Voice: 312-567-3765
Fax: 312-567-8976
E-mail: vcr@iit.edu
Internet: http://agents.iit.edu/vcr

References

1. Genesereth, Michael and Steven Ketchpel, "Software Agents," *Communications of the ACM*, July 1994, pp. 48-53.
2. Maes, Pattie, "Agents That Reduce Work and Information Overload," *Communications of the ACM*, July 1994, pp. 30-40.
3. Wooldridge, Michael and Nicholas Jennings, "Intelligent Agents: Theory and Practice," *Knowledge Engineering Review*, Vol. 10, No. 2, 1995, pp. 115-152.
4. Franklin, Stan and Art Graesser, "Is It an Agent, or Just a Program?" Proceedings of the Third International Workshop on Agent Theories, Architecture, and Languages, Springer-Verlag, 1996.
5. Talukdar, S., V.C. Ramesh, R. Quadrel, and R. Christie, "Multi-Agent Organizations for Real-Time Operations," *Proceedings of the Institute of Electrical and Electronics Engineers*, May 1992.
6. Shaw, Mary and David Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, April 1996.

Software Technology Support Center PSP Services

Personal Software Process (PSP) services are available to Department of Defense and other federal organizations from the Software Technology Support Center (STSC). These services include PSP training for an organization's engineers and its leaders. We also offer a PSP adoption service that focuses on helping an organization successfully adopt and use the PSP.

All STSC PSP instructors have been trained by the Software Engineering Institute (SEI), and are SEI-authorized PSP instructors.

For more information, visit our Web site: **http://www.stsc.hill.af.mil/PSPpage**

or call

Jim Van Buren
Voice: 801-775-3017 DSN 775-3017
Fax: 801-777-8069 DSN 777-8069
E-mail: vanburej@software.hill.af.mil
Les Dupaix
Voice: 801-775-5555 ext. 3088 DSN 775-5555 ext. 3088
Fax: 801-777-8069 DSN 777-8069
E-mail: dupaixl@software.hill.af.mil

S T S C

Coming Events

Third IEEE International Conference on Requirements Engineering

Dates: April 6-10, 1998
Location: Marriott Hotel, Colorado Springs, Colo.
Subject: Software requirements engineering-related problems and results, evaluations of promising research and practice.
Sponsors: IEEE Computer Society Technical Council on Software Engineering, Fujitsu, MCI, and INCOSE
Contact: Charlene Rauber-Svitek crs@sei.cmu.edu; Internet: http://www.cs.technion.ac.il/~icre98/

International Information Technology Quality Conference

Dates: April 13-17, 1998
Location: Orlando, Fla.
Theme: "Providing Proven Solutions for the New Millenium"
Keynote Speakers: Phillip Crosby, Tom DeMarco, William Perry, and Howard Rubin
Sponsor: Quality Assurance Institute
Contact: Voice: 407-363-1111; Fax: 407-363-1112; Internet: http://www.qaiusa.com

Tenth Annual Software Technology Conference (STC '98)

Dates: April 27 – May 2, 1998
Location: Salt Palace Convention Center, Salt Lake City, Utah
Co-hosts: Ogden Air Logistics Center Commander and the Software Technology Support Center
Contact: See page 30, this issue.

International Conference on Computer Languages 1998

Dates: May 14-16, 1998
Location: Loyola University, Chicago, Ill.
Sponsors: IEEE Computer Society Technical Committee on Computer Languages, and ACM Special Interest Group on Programming Languages
Contact: http://www.math.luc.edu/iccl98/

Three Dimensions of Process Improvement
Part II: The Personal Process

Watts S. Humphrey
Software Engineering Institute

Part I of this article (CROSSTALK, February 1998) described the Capability Maturity Model®, why it was developed, and how it can help organizations improve their performance. Part II addresses the Personal Software Process (PSP)^{SM}, which shows engineers how to perform their tasks in an effective and professional way. In the final analysis, to have high-performance software organizations, you must have high-performance software engineers working on high-performance software teams. The objective of the PSP is to show software engineers how to use process principles in their work. Part III of this article (April 1998 issue of CROSSTALK) describes the Team Software Process, which shows integrated product teams how to consistently produce quality products under aggressive schedules and for their planned costs.

Moving from "What" to "How"

Although the Capability Maturity Model (CMM) provides a powerful improvement framework, its focus is necessarily on "what" organizations should do and not "how" they should do it. This is a direct result of the CMM's original motivation to support the Department of Defense acquisition community. We knew management should set goals for their software work but we also knew that there were many ways to accomplish these goals. Above all, we knew no one was smart enough to define how to manage all software organizations. We thus kept the CMM focus on goals, with only generalized examples of the practices the goals implied.

As organizations used the CMM, many had trouble applying the CMM principles. In small groups, for example, it is not generally possible to have dedicated process specialists, so every engineer must participate at least part time in process improvement. We kept describing to engineers *what* they ought to do and they kept asking us *how* to do it. Not only did this imply a need for much greater process detail, it also required that we deal more explicitly with the real practices of development engineers. We needed to show them precisely how to apply the CMM process principles.

Because software development is a rich and sophisticated process, we realized a single set of cookbook methods would not be adequate. We thus chose to deal with fundamental process principles and to show engineers how to define, measure, and improve their personal work. The key is to recognize that all engineers are different and that each must know how to tune their practices to produce the most personal benefit.

The SEI's work is supported by the Department of Defense. Capability Maturity Model and CMM are registered with the U.S. Patent and Trademark Office. Personal Software Process, PSP, Team Software Process, and TSP are service marks of Carnegie Mellon University.

Changing Engineers' Practices

Improvement requires change, and changing the behavior of software engineers is a nontrivial problem. The reasons for this explain why process improvement is difficult and illustrate the logic for the PSP.

The problems related to improving the personal practices of software engineers have long interested me, so after I had been at the Software Engineering Institute (SEI) for several years, I looked for someone else to lead the CMM work so I could address this issue. I decided to first demonstrate how process improvement principles could be applied to the work of individual engineers. Over the next several years, I wrote 62 small to moderate-sized programs as I developed as close to a Level 5 personal process as I could devise.

The results were amazing. I became more productive, the quality of my work improved sharply, and I could make accurate personal plans. The next step was to demonstrate the effectiveness of these methods for others. I first tried meeting with engineering groups to describe what I had done and to get them to try it. Despite management support, this was a dismal failure. One laboratory manager even told his people that it was more important for them to use these methods than to meet their project schedules. The engineers all said they would do so, but none of them did. The question was why not?

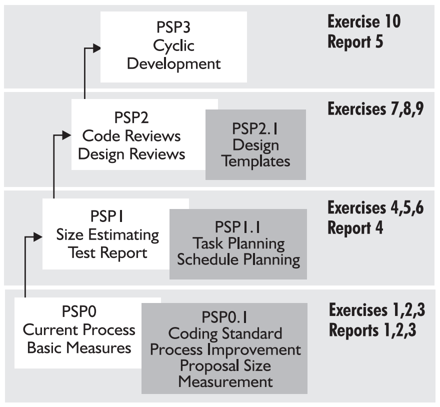


Figure 1. *The PSP process evolution.*

A Question of Conviction

Software engineers develop their personal practices when they first learn to write programs. Since they are given little or no professional guidance on how to do the work, most engineers start off with exceedingly poor personal practices. As they gain experience, some engineers may change and improve their practices, but many do not. In general, however, the highly varied ways in which individual software engineers work are rarely based on a sound analysis of available methods and practices.

Engineers are understandably skeptical about changes to their work habits; although they may be willing to make a few minor changes, they will generally

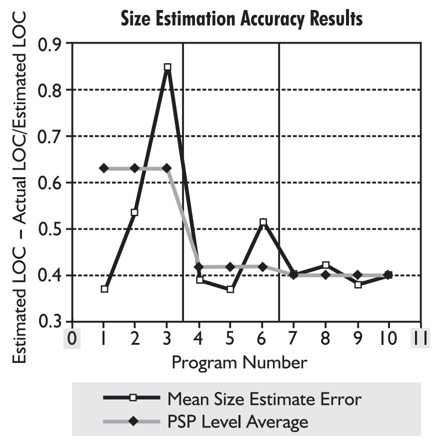


Figure 2. *Size estimation results.*

stick fairly closely to what has worked for them in the past until they are convinced a new method will be more effective. This, however, is a chicken-and-egg problem: engineers only believe new methods work after they use them and see the results, but they will not use the methods until they believe they work.

The Personal Software Process

Given all this, how could we possibly convince engineers that a new method would work for them? The only way we could think of to change this behavior was with a major intervention. We had to directly expose the engineers to the new way of working. We thus decided to remove them from their day-to-day environment and put them through a rigorous training course. As shown in Figure 1, the engineers follow prescribed methods, represented as levels PSP0 through PSP3, and write a defined set of 10 programming exercises and five reports [1]. With each exercise, they are gradually introduced to various advanced software engineering methods. By measuring their own performance, the engineers can see the effect of these methods on their work.

Figures 2 through 5 show some of the benefits engineers experience [2, 3]. Figure 2 shows the average reduction in size-estimating error for nearly 300 engineers who took the PSP course and provided data to the SEI. Their size-estimating error at the beginning of the course is indicated at the left of the chart, and their error at the end of the course is shown at the right. This shows that size-estimating errors averaged 63 percent with PSP0 (the first three programs) and 40 percent for PSP2 and PSP3 (Programs 7, 8, 9, and 10). Note that the PSP introduces a disciplined estimating method (Proxy-Based Estimating) with PSP1 (Program 4) [1].

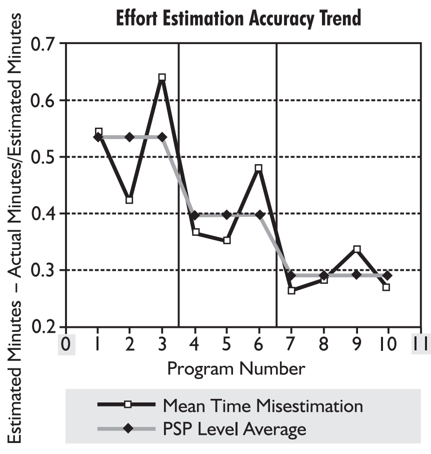
Similarly, for time estimating, Figure 3 shows an improvement from a 55 percent error to a 27 percent error or a factor of about two. As shown in Figure 4, the improvement in compile and test defects is most dramatic. From PSP0 to PSP3, the engineers' compile and test defects dropped from 110 defects per 1,000 lines of code (KLOC) to 20 defects per KLOC, or over five times. Figure 5 shows that even with their greatly improved planning and quality performance, the engineers' lines of code productivity was more or less constant.

Perhaps the most impressive PSP change is in the way the engineers spend their time. With Program 1, as shown in Figure 6, this group of nearly 300 engineers spent on average less time designing their programs than they did on any other task. They even spent more time compiling than designing. At the end of the course, they spent more time designing than in any other technical activity. We have been trying to get software engineers to do this for years. Until they can experience the benefits of more thorough designs, they will likely continue to concentrate on coding, compiling, and testing.

Industrial Results with the PSP

A growing number of organizations are using the PSP, such as Baan, Boeing, Motorola, and Teradyne. Data from some early users clearly demonstrate the benefits of PSP training [4]. Figure 7 shows data from a team at Advanced Information Services (AIS) in Peoria, Ill. They were PSP trained in the middle of their project. The three bars on the left of the chart show the engineers' time estimates for the weeks it would take them to develop the first three compo-

Figure 3. *Effort estimation results.*



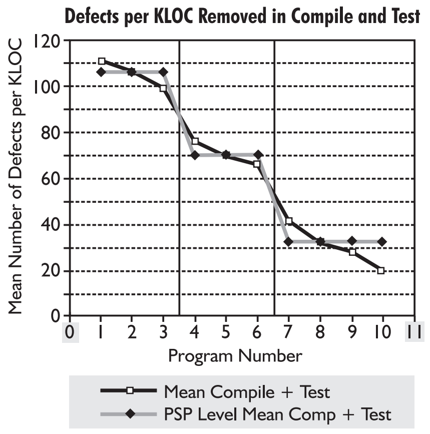


Figure 4. *Quality results.*

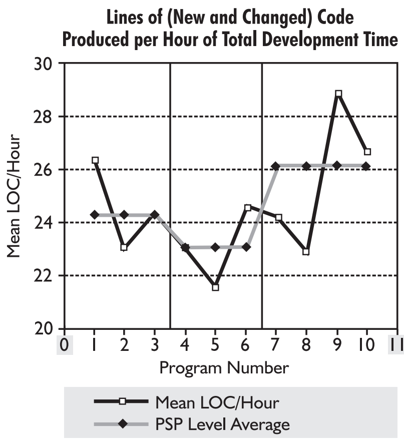
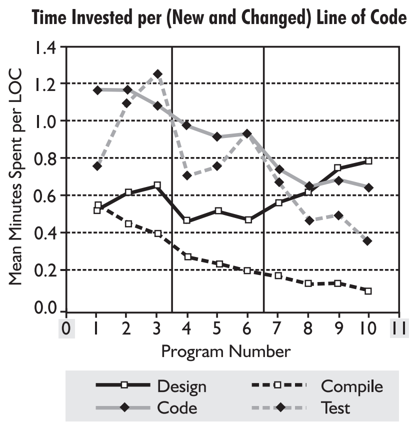


Figure 5. *Productivity results.*

Figure 6. *Effort distribution results.*



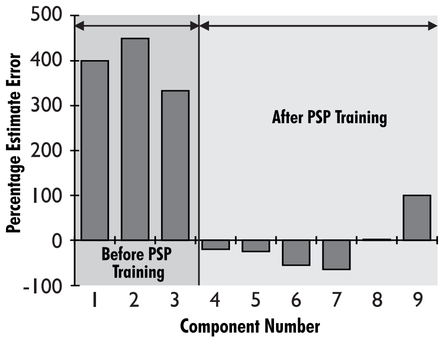


Figure 7. *Schedule estimating error.*

nents. For Component 1, for example, the original estimate was four weeks, but the job took 20 weeks. Their average estimating error was 394 percent. After PSP training, these same engineers completed the remaining six components. As shown on the right, their average estimating error was -10.6 percent. The original estimate for Component 8, for example, was 14 weeks and the work was completed in 14 weeks.

Table 1 shows acceptance test data on products from one group of AIS engineers. Before PSP training, they had a substantial number of acceptance test defects and their products were uniformly late. After PSP training, the next product was nearly on schedule, and it had only one acceptance test defect. Table 2 shows the savings in system testing time for nine PSP projects. At the top of the chart, system test time is shown for several products that were completed before PSP training. At the bottom, system test time is shown for products the same AIS engineers completed after PSP training. Note that A1 and A2 are two parts of the same product, so testing for them was done together in one and one-half months.

Table 1. *Acceptance test improvement.*

Not Using PSP	KLOC	Months Late	Acceptance Test Defects
1	24.6	9	N/A
2	20.8	4	168
3	19.9	3	21
4	13.4	8+	53
5	4.5	8+	25
Using PSP			
1	22.9	1	1

Introducing the PSP

Although the PSP can be introduced quickly, it must also be done properly. First, the engineers need to be trained by a qualified PSP instructor. The SEI trains and authorizes PSP instructors and provides limited on-site PSP training. There is also a growing number of SEI-trained PSP instructors who offer commercial PSP training (see http://www.sei.cmu.edu).

The second important step in PSP introduction is to train in groups or teams. When organizations ask for volunteers for PSP training, they get a sparse sprinkling of PSP skills that will

System test time before PSP training		
Project	Size	Test Time
A1	15,800 LOC	1.5 months
C	19 requirements	3 test cycles
D	30 requirements	2 months
H	30 requirements	2 months
System test time after PSP training		
Project	Size	Test Time
A2	11,700 LOC	1.5 months
B	24 requirements	5 days
E	2,300 LOC	2 days
F	1,400 LOC	4 days
G	6,200 LOC	4 days
I	13,300 LOC	2 days

Table 2. *System test time savings.*

generally have no impact on the performance of any project.

Third, effective PSP introduction requires strong management support. This, in turn, requires that management understand the PSP, know how to support their workers once they are trained, and regularly monitor their performance. Without proper management attention, many engineers gradually slip back into their old habits. The problem is that software engineers, like most professionals, find it difficult to consistently do disciplined work when nobody notices or cares. Software engineers need regular coaching and support to sustain high levels of personal performance.

The final issue is that even when a team of engineers are all PSP trained and properly supported, they still have to figure out how to combine their personal processes into an overall team process. We have found this to be a problem even at higher CMM levels. These are the reasons we are developing the Team Software Process (TSP).

Part III of this article, which describes what the TSP is and how it helps teams to work more effectively, will appear in the April 1998 issue of CROSSTALK. Although the TSP is still in development, early industrial experience demonstrates that it can substantially improve the performance of integrated product teams. ◆

About the Author

Watts S. Humphrey is a fellow at the SEI of Carnegie Mellon University, which he joined in 1986. At the SEI, he established the Process Program, led initial development of the CMM, introduced the concepts of Software Process Assessment and Software Capability Evaluation, and most recently, the PSP and TSP. Prior to joining the SEI, he spent 27 years with IBM in various technical executive positions, including management of all IBM commercial software development and director of programming quality and process. He has master's degrees in physics from the Illinois Institute of Technology and in business administration from the University of Chicago.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
Voice: 412-268-6379
E-mail: watts@sei.cmu.edu

References

1. Humphrey, W.S., *A Discipline for Software Engineering*, Addison-Wesley, Reading, Mass., 1995.
2. Hayes, Will, "The Personal Software Process: An Empirical Study of the Impact of PSP on Individual Engineers," CMU/SEI-97-TR-001.
3. Humphrey, W.S., "Using a Defined and Measured Personal Software Process," *IEEE Software*, May 1996.
4. Ferguson, Pat, Watts S. Humphrey, Soheil Khajenoori, Susan Macke, and Annette Matvya, "Introducing the Personal Software Process: Three Industry Case Studies," *IEEE Computer*, Vol. 30, No. 5, May 1997, pp. 24-31.

Optimizing Software Inspections

Tom Gilb
Independent Consultant

Software Inspections are now widely known to the software industry, but most organizations do not make the most of them. This is because many people misunderstand and misinterpret Inspection. This article aims to provide some direction as to how to get the most value from your Inspections.

Given adequate management support, Inspection can quickly be turned from the initial chaos phase (20 or more major defects per page) to relative cleanliness (two or fewer major defects per page at exit) within a year. To understand Inspection, two points must be clear:

- Inspection, as I define it (shown with a capital "I"), consists of two main processes: the Defect Detection Process (DDP) and the Defect Prevention Process (DPP) [2].
- A major defect ("major") is a defect that, if it is not dealt with at the requirements or design stage, will probably have an order-of-magnitude or larger cost to find and fix when it reaches the testing or operational stages. On average, the find-and-fix cost for a major defect is one work hour upstream but nine work hours downstream.

DDP finds up to 88 percent of existing major defects in a document on a single pass. This alone is important, but DDP actually achieves greater benefit by teaching software engineers. They go through a rapid, individual learning process, which typically reduces the number of defects they make in their subsequent work by two orders of magnitude.

In addition, DDP can and should be extended to support continuous process improvement. This is achieved by including the associated DPP, which is capable of at least 50 percent (first year, and first project used on) to 70 percent (second or third year) defect frequency reduction, and over 90 percent in the longer term. It has also shown at least a 13-to-1 ratio return on investment (ROI). It is the model for Software Engineering Institute (SEI) Capability Maturity Model (CMM) Level 5.

Raytheon provides a good case study. In six years (end 1988 to end 1994), using DDP combined with DPP, Raytheon reduced rework costs (costs of dealing with preventable errors) from about 45 percent to between 5 percent and 10 percent, and had a 7.7-to-1 ROI. They improved software productivity by a factor of 2.7-to-1, reduced negative deviation from budget and deadlines from 40 percent to near zero, and reduced bug density by about a factor of three [5]. Additional detailed costs and benefits can also be found in [2, 3].

Improving Inspections

Following are some key tips about how to improve your Inspection process and how to begin to achieve the kind of results cited above.

Know Your Purpose for Using Inspection

• *Don't misuse Inspection as a "clean-up" process. Use it to motivate, teach, measure, control quality, and to improve your processes.* Most people seem to think Inspection is for cleaning up bad work, bugs, and defects. However, the greatest payback comes when Inspection improves future work, i.e., reduces defect injection. Ensure that your Inspection process fully supports the aspects of *teaching* and *continuous process improvement.*

For continuous process improvement, integrate DPP into conventional Inspections (see [2] for specific details). DPP needs to be practiced early. CMM Level 5 is too important to be put off until later—you need to do it from the beginning.

• *Plan Inspections to address your Inspection purposes.* I have listed over 20 distinct purposes for using Inspection, e.g., document quality, removing defects, job training, motivation, helping a document author, improving productivity, and reducing maintenance costs [3, 5]. Each Inspection will address several of these purposes to varying degrees. Always be aware which purposes are valid for your current Inspection, and formally plan to address them, e.g., by choosing checkers with relevant skills and giving them appropriate checking roles.

Measure Inspection Benefits

• *Measure your benefit from using Inspections.* Inspection should always be highly profitable, e.g., 10-to-1 ROI. If not, it is time to adjust the Inspection process or stop it. Benefits to be measured include rework costs, predictability, productivity, document quality, and ROI [1].

Make Intelligent Decisions on What You Choose to Inspect

• *Use Inspection on* any *technical documentation.* Most people think Inspection is about source code inspection. It *started* there in the early 1970s; however, once you realize that Inspection is not (in the medium term) a clean-up process, you can use it to measure and validate any technical documentation—even technical diagrams.

• *Inspect upstream first.* By the end of the 1970s, IBM and Inspection-method founder Michael Fagan recognized that defects and the profitable use of Inspection lay upstream in the requirements and design areas. Bellcore found that 44 percent of all bugs were due to defects in requirements and design reaching the programmers [4]. If your systems start with contracts, management and marketing plans, you must start your inspection activity there, where the problems start.

One of the most misunderstood dictums to come from the early inspections was "no managers." Wrong! That was in the days when Inspection dealt only with source code. Management inspections may be the most useful you will ever do. I have extremely positive experiences using Inspection with top managers on contracts, marketing, and product development plans.

• *Check the significant portions of the material—avoid checking commentary.* Most organizations waste time checking nonsignificant document areas that do not translate into a final product; defects in such areas cannot trigger major consequences. The result of this indiscriminate checking of trivia, at an optimum rate, is 90 percent minor defects and a 90 percent waste of time. It is like checking comments for 90 percent of the time instead of real code.

We have found that it pays to have a general technical documentation rule that technical authors must distinguish between text (or diagrams) that can translate to serious downstream costs (noncommentary or "meat") and less-important areas (commentary or "fat"). This distinction can be done, for example, by using italics for the fat. Some of our clients have even created Microsoft Word macros to count the volume of noncommentary text and print it on the first page. Of course, the checker is allowed to scan and reference the commentary words but is not obligated to check them against all sources, rules, and checklists. It is not worth it.

• *Use sampling to understand the quality level of a document.* It is neither necessary nor desirable to check all pages of long documents. Representative samples will probably tell you whether a document is clean enough to exit at, for example, 0.2 majors per page maximum remaining.

The main purpose of Inspection is *economic*—to reduce lead time and people costs caused by downstream defects—not primarily to clean bugs. As in Harlan Mills' IBM "Cleanroom" method, bugs should be cleaned up or avoided using disciplines such as Watts Humphrey's Personal Software Process (PSP), Structured Programming, and Continuous Improvement and Verification. If all this works as it should, cleaning is unnecessary, and sampling tells you if it is *economically safe* to release the document. Perfection is not required—it costs infinite resources and is dangerous as a guiding concept.

• *Inspect early and often while a document is being written.* Inspection after a large (100 pages or more) technical document has been "finished" is a common, bad idea. If the process that generates the document is faulty, discover it early and put it right. This saves time and corrects bad processes before they damage your schedule too much.

Focus on Finding the Majors

• *Check at your organization's optimum rates to find major defects.* This is the big one! Most everybody, including so-called teachers of Inspection, manage to miss this point. Or worse, their suggested checking rates are 10 times optimum speed. Optimum *checking* rate is not optimum *reading* rate. Checking in real Inspections involves checking a page against all related documents. This can involve up to 10 or 20 source documents of large size, checklists, and standards. You have to check a *single* line against *many* sources, and it takes time.

Adequate Inspection statistics can prove your employees have a clear, dramatic, and consistent optimum checking rate on specific document types. This ranges between 0.2 and 1.8 pages of 300 noncommentary words per checking hour. At Raytheon, it was about 20 plus or minus 10 lines per hour (0.3 pages). Unfortunately, in spite of their own data, Raytheon suggested rates of about 100 to 250 lines per hour. This was probably because they had finite deadlines and did not understand sampling.

As the checking speed moves toward an optimum speed for effectiveness of finding major defects, the curve for optimum checking rate moves dramatically upward in terms of major defects identified per logical page. The optimum may seem slow, but considering the amount of checking you have to do, it is fast. The main point is that there is a best speed at which to check, and you will be operating at low checking productivity if you fail to heed it.

Note that the optimum checking rate applies both to the checking carried out *before* and to the optional checking carried out *during* the logging meeting. This second check will produce roughly an additional 15 percent defects. You do not need this extra checking if the document is found clean enough to exit as a result of initial checking sampling or if it is so polluted that you have to do a major rewrite anyway.

• *Define a major defect as "possible larger costs downstream."* It does not matter if a defect is not precisely a bug or if it is visible to a customer. If it *can* potentially lead to significant costs if it escapes downstream, classify it as a major and treat it with due respect.

You can help people identify majors by using checklists that specify how to find them. (Note: checklists are only allowed to help interpret the "rules," which are the official standards for writing a given document, and which define defects). Use symbols: "M" for major and "m" for minor after the checklist questions or rule statement. I also often find it useful to use "S" for super major or showstopper (a defect where the downstream effect could be an order of magnitude bigger than an average major). Super majors can be highlighted for management attention.

• *Log only major defects.* This helps avoid the "90 percent minor" syndrome that often hampers Inspection. Employees waste time identifying 90 percent minor defects unless strongly redirected. There are 18 tactics that shift your focus from minor to major defects ([2], pp. 75-76). For example, allow only ideas for finding majors onto the rules and checklists, log only majors at a meeting, and calculate ROI for Inspections only based on majors. A clear message must be given to not waste time on minor defects.

Apply Good Practice When Leading Inspections

• *Use serious entry conditions, e.g., numeric quality of sources.* We are in such a big hurry to waste our own time. Many do not have the discipline to set up and respect entry conditions that prevent wasting time—but you must.

One of the most important entry conditions is to mandate the use of upstream source documents to inspect a "product" document. It is a mistake to try to use the experts' memory abilities instead of updated, Inspection-exited source documents. It is a farce to use source documents with the usual uncontrolled, uninspected, unexited, 20 or more major defects per page to check a product document. It is silly to allow a product document author to use a bad quality source document to generate a product. Inspection does not have to repeat that silliness.

You can ascertain the state of a source document through inexpensive sampling. A half day on a few pages is a small price to pay to know the state of a document that could destroy the quality of all your work and your project.

Another serious entry condition is to do a cursory check on the product document and return it to the author when it is anything less than a quality piece of work bursting at the seams to exit (based on a cursory check that reveals few remaining defects). For example, if while planning the Inspection, the team leader performs a 15-minute cursory check that shows up a few major defects on a single page, it is time for a word with the author in private. Pretend the document was never seriously submitted. Certainly do not waste the time of your team to confirm shoddy work.

In short, learn which entry conditions you need to set, then take them seriously. I would like to see management lead by understanding the importance of this instead of ignorantly thwarting engineers' attempts to do reasonable work by insisting Inspections proceed when the entry conditions have not been met.

• *Make sure you have excellent standards to identify defective practices.* Inspection requires that good work standards be in place. Standards provide the rules for the author when writing technical documents and then for the Inspection process to subsequently check against. Standards must be built by hard experience; they need to be brief and to the point, be monitored for usefulness, and must be respected by the troops. They must not be built by outside consultants or dictated by management. They must be seen as the tool to enforce the necessary lessons of professional practice on the unwary or unwilling.

• *Check against source and kin documents; check them for defects, too.* Because of potentially poor quality control practices and craftsmanship, and because Inspection is imperfect on first pass (30 percent to 80 percent effective), focus on major defects that persist in *source* documents used to produce the document under Inspection, and in kin documents derived from the same source documents. For example, if a functional specification was the product document requiring Inspection, there should be a requirements document as one of the source documents and a testing document as one of the kin documents.

Most people overfocus on the product document, i.e., the document you are inspecting and evaluating for exit. You should probably be finding 25 percent of your total defects *external* to the product document.

• *Use the optimum number of people on a team to serve the current purpose of Inspection, e.g., effectiveness, efficiency, and training.* For 13 years, one large U.S. telecommunications company had 12 to 15 people on each inspection team because each "had" to be there to protect territorial interests. There seemed to be no motivation to cut these costs.

The number of people who are needed on a specific Inspection team is a function of your purposes. Monitor the results of varying team sizes to discover your optimum. If you measure your own Inspection experiences, you will find that effectiveness at finding major defects uses four to six people, efficiency (effect over cost) needs two to four people, and only teaching as a purpose justifies larger numbers.

• *Allocate special defect searching roles to people on the team.* Each person on an Inspection team should be finding *different* defects. Much like a coach on a ball team, the Inspection team leader should assign specialist roles to team members, e.g., identify time and money risks, check against corporate standards for engineering documentation, and check security loopholes.

• *Use checking data (such as pages checked, majors found, time used, and checking rate) from individual checkers to decide whether it is worth holding a logging meeting.* Older types of inspection plunge into the logging meeting without forethought, and consequently waste a lot of time. We have developed a process of logging meeting entry evaluation before going ahead with the logging meeting. We collect data from checkers about checking rates and major-issue density. (To avoid personal conflict, issues—not defects—are logged during the logging meeting. An issue may or may not become a defect.) Based on this data, we make a series of decisions about the logging meeting. Most critical is whether a meeting is necessary. Other decisions include whether to log minors, whether to continue checking, and what is the likely optimum checking rate.

• *Use the individual checkers' personal notes instead of proper meeting defect logs when the major issue volume is (nonexit level) high, or when there is a large number of minor defects.* Checkers should not be required to use any particular method to make notes during checking. But most of them mark a paper document (some use an electronic document) with an underline or circle, or they highlight offending words. It is important that they also note, against offending words, *exactly* which rule was broken (the issue). To note major or minor is less important if all issues found are majors.

Whenever there is a higher volume of issues than would indicate allowable exit, you can happily, with author agreement, return these "scratchings" to the author rather than pedantically log them like good bureaucrats. Authors need to rewrite and resubmit in these cases, using this information to correct their work processes (usually to take sources and rules more seriously).

• *At logging meetings, avoid discussions and avoid suggesting fixes.* Inspection is not for talkers and quibblers—it is for professionals committed to making maximum, meaningful progress on the project. You can have a good time, but not by idle gossip and insults. You are there to measure, not to wear each other out, or get drowned in unprofitable bureaucratic games.

• *Use serious exit conditions, e.g., "maximum probable remaining major defects per page is 0.2 for exit."* Exit conditions, if correctly formulated and taken seriously, can be crucial. It is ridiculous and sad to have the customary "vote" to accept a document when the logged defects are fixed—this completely ignores the known factor of *remaining unfound defects*, which is computable and verifiable from past data and experience.

Remember that Inspection processes, as other testing processes, have a maximum effectiveness for a single pass in the range of 30 percent to 88 percent of existing defects. If the maximum probable remaining defect density is a high-quality low count, e.g., 0.2 majors per page, it matters little if the detected defects are removed; the document is clean enough (economically speaking) to exit without fixing those identified.

If defect density is high, e.g., 20 or more majors per page (quite common), undetected defects, at say 50 percent effectiveness, are more than enough to make exit uneconomical. Ten majors remaining per page in a 100-page document would result in 9 x 10 x 100 hours of additional project work to clean them up by testing and discovery in the field. It costs an order of magnitude less to find them now. Admittedly, it is only the lesser of two evils—you should wish to *prevent* them using DPP rather than have to *clean them up* early using DDP.

Management must understand the large-scale economics of this, making clear policy about the levels of major defects per page that will be allowed to escape ([2] pp. 430-431). The consequences of poor policy here should be deducted from management's pay!

Ensure You Have Provided Adequate Training and Follow-up

• *Give Inspection team leaders proper training, coaching after initial training, formal certification, statistical follow-up, and if necessary, remove their "license to inspect."* Proper training of team leaders takes about a week (half lectures and half practice). Formal Inspection team leader certification (an entry condition to an Inspection) should be similar to that for pilots, drivers, and doctors—based on demonstrated competence after training.

Leaders who will not professionally carry out the job, even if it is because their supervisor is pressuring them to cut corners, need to have their license revoked. If you take professional leadership seriously, your players will take Inspection seriously. Ensure there is an adequate number of trained people to support your Inspections—at least 20 percent of all professionals. Some clients train all of their engineers.

Give Visibility to Your Inspection Statistics and Support Documentation

• *Put your Inspection artifacts on a company Web site.* If you have an Intranet, all relevant Inspection artifacts, standards, experiences, statistics, and problems should be placed on a corporate-wide site as soon as possible.

• *Build or buy an automated software tool to process Inspection basic data.* Use automated software tools to capture summary data and to present trends and reports [7]. Inspection quickly generates a lot of data that is fundamental and useful to managing the process. It is vital that good computer support be given early so the process owners and management take the data seriously and so that early champions are not overwhelmed.

The key distinction between Inspections and other review processes is the use of data to manage them. For example, optimum checking rates must be established early and updated as they change through continuous improvement. It also is vital to statistically see the consequences of inadequate exit levels (too many major defects floating downstream), which then are caught with expensive testing processes. Locally made spreadsheet software is a good start-up tool.

• *Plan Inspections well using a master plan.* We have developed a one-page master plan ([2] p. 401) that goes far beyond the conventional "invitation." We document the many supporting documents needed, assign checkers their special defect-searching roles, and carefully manage rates of checking and the total checking time needed. We establish the formal purpose(s) of this specific Inspection, which vary. We establish a team numeric stretch goal for this Inspection and a strategy to help attain it. A good master plan avoids senseless bureaucracy and lays the groundwork for intelligent Inspections.

Continuously Improve Your Inspection Process

• *Use Defect Prevention Process on the Inspection process.* Finally, recognize that systematic continuous improvement of the Inspection process is necessary. Initially, this is required not only to improve the process but also to learn the Inspection process properly and to tailor it to your organization. ◆

Acknowledgment

I thank Lindsey Brodie for her editing and assistance on this article.

About the Author

Tom Gilb is a freelance consultant. Born in California, he has been a resident of Norway since joining IBM, there, in 1958. He has the Department of Defense as his favorite worthy cause.

E-mail: Gilb@ACM.org

References

1. Raytheon Defense Electronics, http://www.sei.cmu.edu/products/publications/95.reports/95.tr.017.html.
2. Gilb, T. and D. Graham, *Software Inspection*, Addison-Wesley Longman, London, England, 1993.
3. http://www.stsc.hill.af.mil/SWTesting/gilb.html.
4. Pence, J. L. Pete and Samuel E. Hon III, Bellcore, Piscataway, N.J., "Building Software Quality into Telecommunications Network Systems," *Quality Progress*, October 1993, pp. 95-97.
5. The Raytheon Report, http://www.sei.cmu.edu/products/publications/95.reports/95.tr.017.html. Also see http://www.Result-Planning.com and http://www.stsc.hill.af.mil/SWTesting/gilb.html.
6. A set of slides that correspond to this article should be at the Washington, D.C. Spin site under "Old Lectures," http://www.software.org/DCSpin.
7. Software Development Technologies, Software Inspections Automation, Edward Kit, sdt@sdtcorp.com, http://www.sdtcorp.com.

A Software Development Process for COTS-Based Information System Infrastructure: Part 1

Greg Fox, *TRW Systems Integration Group*
Karen Lantner, *EDS*
Steven Marcom, *TRW Information Services Division*

The Infrastructure Incremental Development Process fills two gaps within the realm of software development processes: It provides a programmatic, prototype-driven, but carefully controlled approach to commercial-off-the-shelf selection and integration, and it provides a process that specifically addresses development not of applications but of the infrastructure of a large distributed information system.

The level of abstraction at which the software developer works has changed markedly throughout the last 40 years. Early programmers used ones and zeros to control the electronic switches within computers. That technology was followed by procedural languages that, from the programmer's view, removed much of the physical housekeeping associated with the specific design of the computer. In recent years, an even higher level of abstraction has appeared: the integration of prepackaged commercial-off-the-shelf (COTS) software into system designs. In addition, the domain of software development has become segmented into different layers. For example, application-level software development can be distinguished from infrastructure-level software development.

The Emerging Divide in System Functionality

The value of layering in software architecture and implementation is an established concept. Key to the layering model is the idea that through use of defined interfaces between layers, the impact of changes in any given layer can be largely isolated from the other layers.

The concept of a services layer and of specialized software in the system acting as service providers has continued to grow from the simple beginnings in the operating system to become a fundamental architectural concept in modern system design. As reuse and portability of software applications across different vendor hardware platforms become an increasingly important goal, a more sophisticated model of service layers and service providers has emerged. The open systems movement cites application portability across computing platforms as a major economic driver [1, 2].

The National Institute of Standards and Technology (NIST) Application Portability Profile (APP) [3] provides one convenient model to define system layers and services that support portability. This model, along with standards, helps achieve application portability by guiding designers who plan to code new information systems in their entirety and by guiding selection of available software computing components from those available in the marketplace.

Modern information system design models separate the business-specific application software layer in a system from the technology-based infrastructure software layer. An illustration of this approach is the information engineering method of separating business system architecture from technical architecture, which contains the computing infrastructure [4]. This separation into software layers, which is less formally addressed in other design methods, recognizes that change and evolution in information systems are driven by two independent forces: change in business requirements and change in technology. Decoupling the impact of business rule change from change in technology decreases the total amount of system rework necessary to support system evolvability over time. This decoupling is effectively implemented by modeling the infrastructure software using the concept of services layers and service providers.

Views of Infrastructure

There are two ways to look at the infrastructure. One view is the services view of infrastructure as seen by business application developers. It includes Human Computer Interface, Systems Management, Security, Work-Flow Management, Telecommunications, Data Interchange, Transaction Processing, Data Management, and Operating Systems. This grouping of infrastructure services was derived from the NIST APP. Infrastructure services are delivered to the applications through an application programming interface (API).

The second view is the structural view, which includes the kinds of components infrastructure developers use to construct their view of the infrastructure: a set of connected software, network, and hardware components. These include developed software components, COTS software components, communications circuits, local area networks, special purpose servers, general purpose servers, workstations, and laptops.

An additional set of functionalities, treated as part of the infrastructure during the development process, are the technical applications needed to operate the system. These applications neither implement business functionality nor provide services to the business application. They are, for example, the tools for system security administration, database

© 1997 IEEE. This material is adapted and reprinted, with permission, from a paper presented at the IEEE/SEI-sponsored Fifth International Symposium on Assessment of Software Tools and Technologies, Pittsburgh, Pa. held June 3-5,1997, pp. 133-142. Part 2, which will appear in the April 1998 issue of CROSSTALK, describes real-world applications of the IIDA model and examines the practical lessons learned and pitfalls encountered.

administration, system configuration control, software distribution, and, in general, the tool-set for enterprise-level systems management. Other infrastructure services are also used internal to the infrastructure but are not visible to business applications or end users. For example, a remote data access protocol is a level of service provided between infrastructure components that is used to construct a mechanism to access data: It is not directly visible to business applications or end users.

Infrastructure services provide functionality that the application developer can access external to the application and, therefore, does not develop as part of the application. Economy of scale is achieved through common use of technical services by application development projects across the enterprise. Programmers can access infrastructure services without regard to how underlying infrastructure services have been implemented using a properly designed API. By allowing application and infrastructure development to be separate and independent, infrastructure enhancements, e.g., increased performance, additional services, and new computing platforms, can be made with minimal effects on application development.

The COTS Challenge for Infrastructure

Although distributed systems (popularly described as client-server or networked systems) dominate today's computer system design, they still have the character of adolescence. We are in the middle of a dramatic and somewhat uncontrolled expansion and evolution of standards for COTS software products for distributed systems. COTS products provide portions of needed supporting technical functionality to turn collections of computing platforms into unified, distributed computing environments. Available COTS software products offer varying degrees of standards compliance, interoperability, heterogeneous computing platform support, security functionality, performance efficiency, and distributed environment transparency for applications using their services.

Two separate panels at the 1995 Software Engineering Institute/Microelectronics and Computer Technology Corporation Symposium on "The Use of COTS in Systems Integration" concluded that "there is a need for process definitions for COTS usage," [5] and "new lifecycle models for COTS integration projects are needed." [6] Currently, documented software development lifecycle processes provide little practical guidance to developers to achieve the advantages of COTS software or to assist in the selection of specific products from the myriad available. COTS product selection and integration are complicated by an intrinsic set of special characteristics: incompatibility, inflexibility, complexity, and transience.

Development Lifecycle Process Impact

The special characteristics of COTS software integration change the emphasis in the classic waterfall lifecycle stages of planning, definition, analysis, design, construction, integration and test, implementation, deployment, and maintenance. COTS-based development differs from business application-oriented development in that the COTS selection process must occur early in the lifecycle. COTS evaluation and selection become a critical part of the early analysis process rather than a peripheral activity within the later design process. The challenges of COTS incompatibility, inflexibility, complexity, and transience must be addressed in the selection process because the infrastructure will ultimately consist of a suite of COTS products that must operate in harmony.

In addition, since COTS software does not require coding but does require integration with other components, it starts the lifecycle as a partially developed component. The design, construction, and integration and test development stages must be recast to accommodate early COTS software integration and testing as well as to develop glue code: interface software, configuration files, scripts, utilities, and data files required to make the COTS software deliver its intended functionality. The proper development and testing of the glue code to make a COTS package work may not be a trivial undertaking. For more complex COTS software, the development of glue code might need to be treated in the same manner as the development of a traditional custom-coded software module.

When a COTS product enters the development process, the first task is to test and integrate it into the system. This activity starts early in the development process. Waiting until late in the development process to test and integrate COTS products, particularly those that are complex, will not give adequate time to master all their intricacies and complexities. COTS product testing and integration activities must be interwoven into more of the development process stages.

COTS Incompatibility

Many vendors do not develop their products along the lines of the layering models discussed earlier. As this is being written, no single commercially available software product or product family can provide all the infrastructure services needed for an enterprise-level information system of substantial size or complexity. The problem to be solved in system design and development is to select a compatible set of software products that can be integrated together and augmented by glue code to produce a complete set of services.

In an ideal world, a set of products that provide all the needed infrastructure services would simply "snap together" like the pieces in the puzzle shown in Figure 1. In the real world, this is not the case: when put together, COTS pieces have gaps and overlaps. At any point in time, the set of services that a system designer can specify as useful exceeds what is available in mature products in the marketplace. The resulting gaps can be overcome in two ways. One is by traditional design and development of custom infrastructure software added around the commercially available products selected (either adding layers between the COTS-based infrastructure and the applications or adding custom service-provider software that is concep-

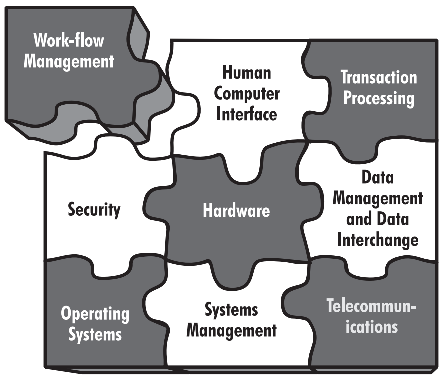


Figure 1. *Generic information system infrastructure service.*

tually parallel to the COTS software). Another way is by leaving it to the application designers to deal with at the application level.

Overlaps between products can cause a greater system design problem than gaps. Commercial software suppliers are driven much more by a desire to capture larger segments of the marketplace than they are by adherence to recommended system implementation layering models. For example, boundaries between database access, transaction processing, and work-flow management software products begin to overlap and blur as each vendor community expands its product's features in pursuit of increased market share.

This expansion in features is driven by requests for increased functionality by the installed base, not by the boundaries defined in layering models. The net result is that certain products and product sets do not work synergistically with other products; yet, none of the products on its own is complete enough to provide all of the necessary functionality. Selection of a specific product that provides a certain set of services often precludes selection of another functionally complementary product.

COTS Inflexibility

The inflexibility characteristic of COTS software can cause both design and integration difficulties. Unlike custom-developed software, when a piece of commercial software exhibits a behavior not expected by the system designer, the developer cannot merely change the behavior of that software but must either replace the software, work around the unexpected behavior, or change requirements. Understanding the behavior of an unmodifiable software component is a different process than specifying the behavior of a component to be constructed. Most documented software development methods take the latter approach and do not address the former.

COTS Complexity

The complexity characteristic of many of today's advanced COTS software products causes distortions in the traditional development process time line. The flexibility and tailorability of product families like transaction monitors, work-flow managers, and system management frameworks mean a significant education investment. The investment must be made upfront before the product can be fully evaluated for selection, and in cases when the product proves unsuitable, the investment might have a zero net return. Experience shows that the selection process for one major product can require three to six months of calendar time, multiple engineers and programmers, access to sophisticated suites of hardware and software environments, and will likely entail the purchase of vendor-provided training classes.

The more complex COTS software products are tailorable and scalable to multiple hardware configurations, software environments, and workload environments. To achieve this flexibility, they contain from dozens to hundreds of adjustable parameters (or "knobs"). Each of these must be set for the specific system configuration. If the system is being built for deployment in multiple locations with different hardware configurations or workload environments, the COTS software parameters might need to be tuned for each installation. This can be a complex task requiring product expertise, experience with the behavior of the integrated system, and, potentially, support from analytic modeling efforts. Software configuration files for each location might need to be tailored using the information developed during system integration. Not only does this require additional development effort, the scheduling process must recognize that just because the system has been integrated and tested in a test facility does not mean that it can be quickly made operational at a production location. The tailoring and tuning process for each location's configuration can require days, weeks, or months.

COTS Transience

COTS software products are characterized by periodic updates. Updates might add functionality but are often incompatible with other system components. On the other hand, remaining with older versions of COTS products might cause future interoperability problems with upgrades to other COTS software. COTS software updates, particularly operating system updates, must always be evaluated for insertion into the system, since critical vendor maintenance and support for older versions often ceases. Management, cost, and technical factors in the transition to new COTS software versions can be formidable, particularly in a system with dozens of interrelated products upgraded by their vendors on different calendar cycles.

The Infrastructure Incremental Development Approach (IIDA)

The development of a COTS-based technical infrastructure demands an approach that is fundamentally different from traditional approaches used for business-oriented applications: one that is heavily prototype-oriented, emphasizes testing, and evolves through multiple iterations. The IIDA is a tailored lifecycle that preserves the benefits of existing structured processes for software development while adapting to the particular characteristics of integrating COTS products. The IIDA is a combination of the classical waterfall development model [7] and the spiral development model [8], but the emphasis is on establishing compatibility and completeness rather than on component-level specifications.

Overview of IIDA

The IIDA is an iterative and incremental approach to infrastructure development where each version of the infrastructure is an increment that is integrated into

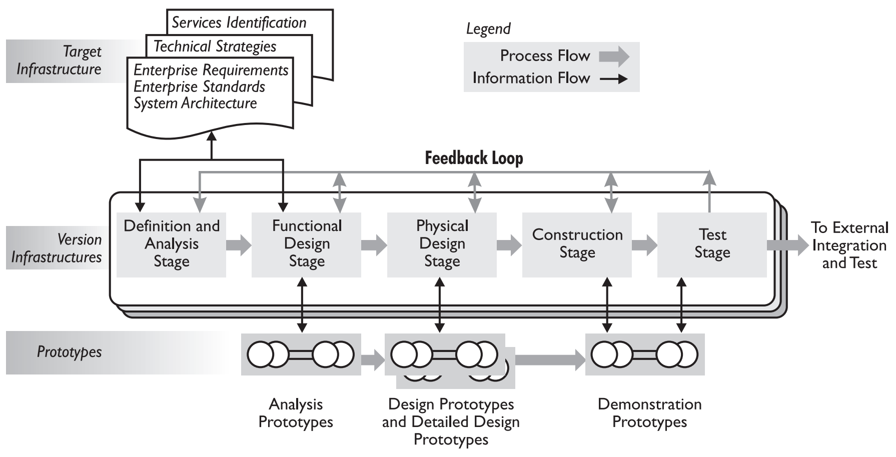


Figure 2. *Infrastructure development approach.*

the existing infrastructure baseline. Within each version, development proceeds in time-sequenced stages with iterative feedback to preceding stages (see Figure 2).

The target infrastructure is the long-term vision for the infrastructure. It is defined and subsequently refined during the Definition and Analysis Stage through a top-down process of analysis of the enterprise requirements, enterprise adopted standards, and the system architecture. The Technical Strategies component captures the high-level description of the complete system vision and defines how the infrastructure will operate [9]. The Services Identification component is built up over time and is influenced by technology trends, product assessments, and the anticipated needs of the business applications.

Development cycle stages are augmented with a series of structured prototypes for COTS product evaluation and integration. For each COTS family, the prototypes evolve from initial analysis prototypes for a make-or-buy decision to, first, a series of design prototypes for COTS product selection and detailed assessment, and next, to a demonstration prototype that becomes part of the development test bed. The timing of the prototypes aligns with the development stages, which depend on the products from their corresponding prototypes. This close coupling of prototyping and classic development stages characterizes the IIDA.

Each pass through the stages in Figure 2 yields an incremental version of the infrastructure that can be integrated with applications and deployed. After the implementation of each version, successive developmental cycles are initiated. The infrastructure thus evolves toward the target infrastructure by providing an increased level of services to business applications and developers and by incorporating new underlying technology and products.

Infrastructure components are integrated into the existing infrastructure baseline. The components in this integrated infrastructure baseline are then ready to be integrated and tested with business applications. Infrastructure development ends with a technical platform upon which business applications can run effectively rather than with an operational product. The scope of this article is infrastructure development: It does not include the external integration, testing, or distribution of business applications.

IIDA Stages

The following is a summary of the major activities of each IIDA stage.

Definition and Analysis Stage

- Enterprise requirements and standards, system architecture, and technical strategies are defined and refined.
- Version-specific functional infrastructure requirements are established by considering business application areas, architectural imperatives, and technology availability.

Functional Design Stage

- Services included in the target and current versions are identified and defined.
- Prototypes are used to identify leading candidate COTS components.

Physical Design Stage

- Interfaces between applications and infrastructure are defined (API is established).
- Internal design of services is defined both functionally and technically.
- COTS and to-be-built components are identified.
- Prototypes are used to select and characterize COTS components.
- Preliminary bill of materials is created for acquisition of equipment and COTS software products.
- Design is calibrated for scaling and performance considerations to provide site designers with site configuration guidelines.
- Structure of each to-be-built component and its interfaces is defined.

Construction Stage

- To-be-built components are constructed.
- Glue code is developed, and the unit is tested.
- COTS components, glue code, and built components are integrated into the infrastructure using the demonstration prototype as a test bed.

Test Stage

- Infrastructure versions are tested prior to being integrated and tested with business applications.

IIDA Milestones and Deliverable Documentation

The infrastructure development approach uses formal and informal reviews, turnovers, and walk-throughs to maintain the degree of formality necessary to control and communicate the design (see Figure 3). Formal reviews include

- Technical Review at the end of the Analysis Stage.
- Design Review during the Physical Design Stage.
- Test Review at end of the Test Stage.

Formal reviews are attended by organizations external to the infrastructure development group as well as infrastructure developers and managers.

These reviews occur once during the development cycle for each version of the infrastructure.

Other reviews, turnovers, and walk-throughs are informal, rolling peer, or management reviews that typically occur when pieces of the design, construction, or integration are ready to be walked through. Infrastructure developers and managers participate in the following informal internal reviews.

- Top-Level Design Walk-throughs during the Definition and Analysis Stage.
- Design Turnovers (from design to development organization) during the Physical Design Stage.
- Detailed Design Walk-throughs at the end of the Physical Design Stage.
- Code Walk-throughs during the Construction Stage.
- Test Design Walk-throughs during the Construction Stage.
- Development Turnovers (from development to test organization) at the end of the Construction Stage.

The lower portion of Figure 3 shows the key documents produced during the IIDA process. Target infrastructure documents, which include Enterprise Requirements, Technical Strategies, and Services Identification, are created once at the beginning of infrastructure development and updated as versions are produced. Version-specific infrastructure documents are created for each infrastructure version. Not shown in the tables are the informal documentation packages developed for the formal reviews and informal walk-throughs.

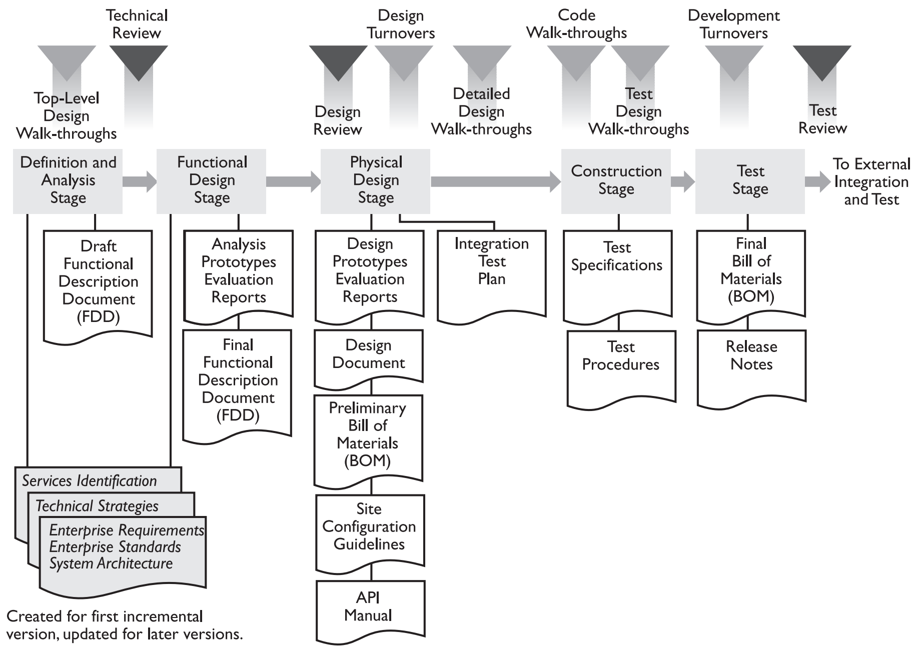

Technical Review
Design Turnovers
Code Walk-throughs
Development Turnovers
Top-Level Design Walk-throughs
Design Review
Detailed Design Walk-throughs
Test Design Walk-throughs
Test Review
Definition and Analysis Stage
Functional Design Stage
Physical Design Stage
Construction Stage
Test Stage
To External Integration and Test
Draft Functional Description Document (FDD)
Analysis Prototypes Evaluation Reports
Final Functional Description Document (FDD)
Design Prototypes Evaluation Reports
Design Document
Preliminary Bill of Materials (BOM)
Site Configuration Guidelines
API Manual
Integration Test Plan
Test Specifications
Test Procedures
Final Bill of Materials (BOM)
Release Notes
Services Identification
Technical Strategies
Enterprise Requirements
Enterprise Standards
System Architecture
Created for first incremental version, updated for later versions.

Figure 3. *Infrastructure milestones and deliverable documents.*

The Critical Role of Prototypes

At the heart of the IIDA approach is a series of tailored prototypes, shown as Analysis, Design, Detailed Design, and Demonstration prototypes in Figure 2, which also illustrates their respective time phasing in the overall process. This can be viewed as a tailoring of the spiral development model as each successive set of prototypes narrows the solution space for the final implementation.

Analysis Prototypes

Analysis prototypes are used to identify leading candidate COTS software products in each COTS family. A COTS family is defined as a group of COTS software products that performs similar functions or provides related services to the application developers. Analysis prototypes are designed to exercise a COTS product to determine its general capabilities and to discover how well it satisfies the needs of the current version of the infrastructure. Selection of the best product in each family is performed later using the design prototypes.

A sample application can be written to serve as a test vehicle for the family of products under evaluation because infrastructure, by its very nature, provides services rather than active applications. The results of the analysis prototypes feed the version-specific services definition and the version-specific services application programming interface (API) efforts with information on available COTS product behavior and performance. A suite of COTS products will be recommended as a result of these prototypes that, when combined with custom-developed glue code and to-be-built software, cover all the requirements of the combined service areas.

Analysis prototypes are also used to examine emerging technologies for possible inclusion in future versions of the infrastructure. Technology insertion plays an important role in infrastructure evolution from version to version.

Through the analysis prototypes, methods to implement target technical strategies into future infrastructure versions can be postulated and developed. In this role, the analysis prototype supports the evolving definition of the long-term vision or target infrastructure.

Design Prototypes

Design prototype help select the best COTS product to incorporate into the design from several candidates in each area identified through the earlier analysis prototypes. A design prototype exercises a COTS product to determine its functional capabilities and how well it performs in accordance with its documentation. Specific benchmarks can be run in addition to functional tests. Sample applications will usually be written as test vehicles for the products under evaluation and to stimulate service performance under conditions that would be found in the application environment.

Detailed Design Prototypes

Detailed design prototypes are a special case of the design prototypes. They serve as proof-of-concept prototypes and are designed to exercise the selected COTS products to demonstrate that detailed

COTS product capabilities are consistent with the design expectations. Sample applications are usually written to serve as a test vehicle for the products under evaluation. The results of the detailed design prototypes feed the services' detailed internal design with information on COTS behavior and performance and with specific language and syntax requirements to invoke services.

At this level of detail, designers might find that a COTS product does not perform as documented or as expected or that there are unexpected side effects of a product's behavior. The functional design activity receives this feedback, and may need to modify or redesign the solution with a substitute COTS product. The evaluation documentation created during earlier analysis and design prototypes is used to streamline alternate COTS selection.

Demonstration Prototype

The demonstration prototype is used to unit test infrastructure components and to serve as a platform for infrastructure component-to-component integration. The sample applications used for the design prototypes might be reused if they are robust enough to exercise the elements tested in the unit test.

The results of the demonstration prototype feed back into the unit test activity. Unlike the analysis and design prototypes, which are investigative and throwaway in nature, the demonstration prototype is cumulative and evolves into a test-bed environment for the infrastructure.

Application of IIDA

Between 1994 and 1997, the IIDA method was applied to develop the initial versions of an infrastructure to support business application developers for a large enterprise-wide heterogeneous system. In the April 1998 issue of CROSSTALK, we will describe the application of the IIDA model to that development and examine the practical lessons learned and pitfalls encountered. ◆

About the Authors

Greg Fox is a TRW Systems Integration Group technical fellow and the director of technology for the Information Services Division. He has 28 years experience in mostly large or complex information systems. He has lead the architecture development and system integration for several large COTS-based systems and has been TRW's information systems infrastructure project manager and chief architect for the Integration Support Contract for Internal Revenue Service (IRS) modernization. He has engineering degrees from Massachusetts Institute of Technology and University of Southern California and has published over a dozen papers.

TRW, Inc.
MVA1/4943
12900 Federal Systems Park Drive
Fairfax, VA 22033
Voice: 703-876-4396
E-mail: greg.fox@trw.com

Karen W. Lantner is a program/project manager for EDS in New York City. She has 24 years management and technical experience, during which she has managed and consulted on large federal software development and COTS integration projects. A member of the team that developed the EDS Systems Life Cycle Methodology, she continues to have a special interest in software development methods. She has a bachelor's and a master's degree from Brown University.

EDS
A5N-B50
13600 EDS Drive
Herndon, VA 22071
Voice: 800-336-4498, box no. 52032
E-mail: karen.w.lantner@aexp.com

Steven Marcom is a senior systems analyst with the Information Services Division of TRW. He has 30 years managerial and technical experience developing computer systems for civil government, defense, and commercial customers. He was TRW's systems lifecycle deputy manager and information systems infrastructure process engineer for the Integration Support Contract for IRS modernization. He has been active in Rapid Application Development, COTS integration, and prototyping activities. He has a bachelor's degree from Pomona College and a master's degree from the American University of Beirut, both in mathematics. He teaches software development and integration at TRW.

TRW, Inc.
FP1
12900 Federal Systems Park Drive
Fairfax, VA 22033
Voice: 703-803-4814
E-mail: marcoms@gisdbbs.gisd.trw.com

References

1. Berson, A., "Openness and Proprietary Standards," *Client/Server Architecture*, McGraw-Hill, New York, 1992, Section 1.2.2.
2. Cerutti, D. and D. Pierson, "The Rise of Open Systems," *Distributed Computing Environments*, McGraw-Hill, New York, 1993, Chap. 2.
3. National Institute of Standards, Application Portability Profile, The U.S. Government's Open System Environment Profile OSE/1 Version 2.0, NIST Special Publication 500-210, June 1993.
4. Martin, James, *Information Engineering*, Prentice-Hall, Englewood Cliffs, N.J., 1989.
5. Software Engineering Institute, "A Commercial/Business Perspective," *Proceedings of the SEI/MCC Symposium on the Use of COTS in Systems Integration*, Special Report CMU/SEI-95-SR-007, June 1995, p. 24.
6. Software Engineering Institute, "Systems Architecture and COTS Integration," *Proceedings of the SEI/MCC Symposium on the Use of COTS in Systems Integration*, Special Report CMU/SEI-95-SR-007, June 1995, p. 26.
7. Royce, W.W., "Managing the Development of Large Software Systems: Concepts and Techniques," *Proceedings of ICSE9*, IEEE Computer Society Press, 1987.
8. Boehm, B.W., "A Spiral Model of Software Development and Enhancement," *Computer*, May 1988, pp. 61-72.
9. Cooper, R. and G. Fox, "Technical Strategies to Guide the Design of Distributed Information Systems," *SIG Technology Review*, TRW Systems Integration Group, Winter 1996.

Implementing a Paperless Environment
The NAVSTAR GPS Block IIF Engineering Management System Project

Lon Mehlman
Computer Sciences Corporation

This article describes the rationale and award-winning benefits of a computer-aided acquisition lifecycle support system. The system has produced savings in cost and efficiency, all while boosting oversight capabilities. The robust information systems infrastructure for the NAVSTAR GPS Joint Program Office allows lifecycle logistics support and separates applications from data, thus protecting major investments in databases. The system has allowed a reengineering of business processes by integrating work-flow and document management. The system also establishes an open architecture for future application and process integration through the extensive use of government and industry standards.

The Navigation Satellite Tracking and Receiving (NAVSTAR) Global Positioning System (GPS) Joint Program Office (JPO) is a joint-service, multinational organization with over 375 employees. The office develops, acquires, and sustains a 24-satellite constellation, a worldwide satellite control network, over 80,000 receiver systems, and a nuclear detonation detection system. The system is a priority Department of Defense (DoD) force enhancement program that provides the capability to precisely determine position, velocity, and time and to pinpoint nuclear events.

The JPO is located at four primary sites: Los Angeles AFB, Calif.; Peterson AFB, Colo.; Robins AFB, Ga.; and Patrick AFB, Fla.

In mid-1992, the GPS JPO was faced with a major problem. At that time, the 375 users comprising the program office used numerous PC-based applications to accomplish various tasks. Printers were shared through serial data switch boxes. Computer support consisted of several people traversing the building all day in a futile attempt to "standardize" the software on users' systems and keep the various printers and printer interfaces operational. Systems support was becoming exceedingly difficult and was spiraling hopelessly out of control.

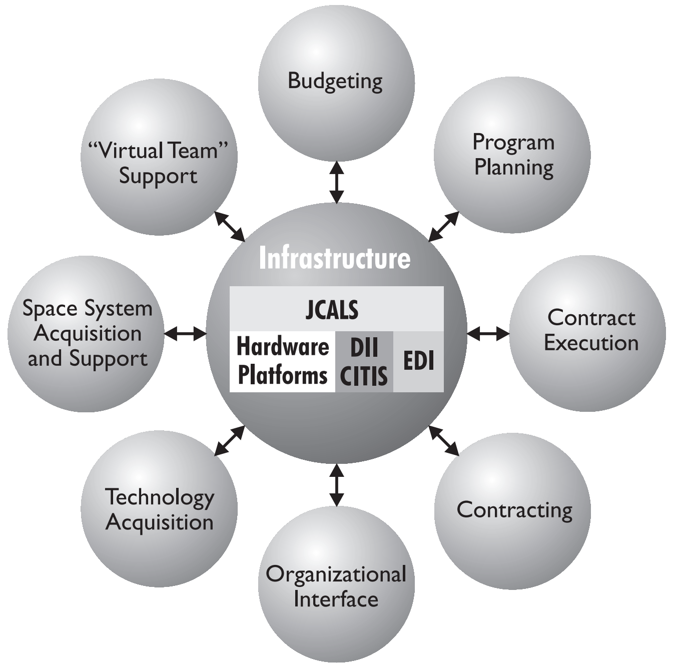
End users would access myriad various mainframe applications to accomplish their job functions. Several proprietary systems hosted on proprietary hardware and operating systems were in place (IBMs VAXs, WANGs, HP 3000s, etc.). Each system and application was its own "island of information." Subsequently, even though there was a physical network in place, there was no communication between systems. Users could not send data from one system to other systems or other users. They had to continue to use paper.

At the same time, the program office continued to generate thousands of pages of paper-based documents and information daily. Air Force leadership was pressing for all program offices to implement acquisition reform initiatives. The program office's leadership was pressing for the introduction of cross-functional integrated product teams (IPTs). This created the need for information sharing among geographically dispersed individuals, the need to open new lines of communication, and the requirement for greater and faster access to all program data.

GPS Engineering Management System

Acquisition reform is a new philosophy for weapons systems procurement that emphasizes government "insight" into contractor processes rather than oversight. The GPS Engineering Management System (GEMS) is a distributed enterprise information infrastructure that is being developed by the NAVSTAR GPS JPO to support Integrated Weapons Systems

Figure 1. *Business processes overlaid upon the GEMS infrastructure.*



Management (IWSM) and acquisition reform initiatives within the JPO. IWSM is an all-encompassing, cradle-to-grave weapons systems management concept. GEMS:

- Allows the integration of JPO business processes with the system development processes of its contractors by taking advantage of the latest advances in information technology.
- Enables the creation of several paperless processes in an integrated GPS program-wide environment, including a cost and schedule management process, a document review process, data call processes, and an engineering change proposal process.
- Separates applications from data, protects major investments in JPO databases, and facilitates business process reengineering by integrating workflow and document management.
- Is an innovative approach to JPO business process automation that combines the DoD Joint Continuous Acquisition and Lifecycle Support (JCALS) system, best-of-breed industry standard commercial-off-the-shelf (COTS) software and hardware, and electronic delivery and access to all unclassified program data to the JPO.

Electronic delivery of data to the JPO is accomplished by the implementation of a Contractor Integrated Technical Information Service (CITIS) for all prime contractors that participate in GPS JPO programs. CITIS is an electronic link between the JPO's GEMS and the information systems used by GPS contractors. CITIS also includes the use of standard data formats, the GEMS shared data service client software, GEMS workstation client software, and other mutually agreed to COTS software tools.

Program data developed by GPS contractors is made available or delivered to the JPO via the electronic link. Documents delivered to the JPO are placed into a shared electronic library that maintains version control, access control, and status of the data. After the data is delivered to the JPO, JPO IPT members start the coordination of the documents electronically by routing program data through the JPO via the JCALS workflow manager.

Objectives

The objectives of the GEMS project are to redesign the GPS JPO's information systems infrastructure to directly support the concepts of IWSM, integrated product development, concurrent engineering, acquisition reform, integrated weapons systems management, and the seamless integration of JPO business processes that can span across the program office and its contractors.

The GPS Block IIF Program, responsible for the procurement of the next generation of GPS satellites, foresaw the critical need for GEMS, and fully supported GEMS objectives.

Lt. Col. Al Moseley, the GPS Block IIF program manager, stated, "The Block IIF program would be the first integrated product team in GPS, and one of the first in the Air Force and the DoD, to implement a paperless system to meet program and acquisition reform objectives."

Implementation for GPS Block IIF

The GEMS infrastructure was implemented in a modular fashion, one process at a time, and rolled out incrementally to each IPT within GPS. Over the past year, GEMS has expanded from a pilot process to receive and review Engineering Change Proposals electronically to one that now allows users to perform all configuration and data management on-line and integrate the cost and schedule management process (see Figure 1).

The GEMS configuration and data management tools integrate and automate the JPO data management process. The data management tools give JPO users the ability to generate AF (Air Force) Form 585 and AF Form 1423, conduct data calls, conduct data scrubs and track all Contract Data Requirement Lists (CDRLs) under review. The tools also make it easier to board documents at the JPO configuration control board and report on data metrics by extracting the required data from the GEMS database.

Acquisition reform calls for a reduction of the number of CDRLs for a program. One of the management principles of the GPS Block IIF Program is electronic access to all unclassified program data. The GEMS data management tools, originally used to determine which CDRLs were to be placed on contract, now help the JPO determine what program data contractors are required to make available electronically via GEMS and CITIS. The use of GEMS allowed the GPS Block IIF Program to reduce the number of CDRLs placed on contract from 339 to three (see Figure 2).

Benefits

GEMS allows the GPS Block IIF and related programs to immediately begin doing things better, faster, and cheaper. In terms of the quality of JPO business processes, measurable improvements have been noted in the following areas.

Shortening the process cycle. Prior to GEMS, the processing cycle for authentication of a system specification was 18 to 24 months; the new authentication process is now six months. The reasons for most delays can be immediately detected via the work-flow and corrective action can be taken.

Standardizing JPO processes. The paper-based processes varied greatly; now, most JPO processes are documented not only in operating instructions but also in GEMS work-flow templates. The work-flow templates show the proper routing of documents and tasks to the proper offices for each type of process. When action is required on an electronically delivered document, an individual in the office of primary responsibility can select the appropriate work-flow process template for a given function, make any necessary adjustments, start a "job," and accurately track the status of the document.

Empowered team orientation. The reengineered GPS Block IIF IPT business processes use GEMS. This results in a largely matrixed organization, grouped by IPTs, in which each team is responsible for a product and given sufficient decision-making authority. In the old system, documents were being circulated

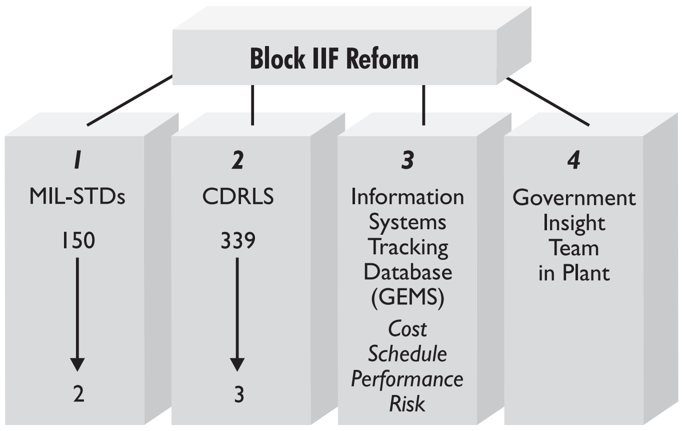

Block IIF Reform
1
MIL-STDs
150
2
2
CDRLS
339
3
3
Information Systems Tracking Database (GEMS)
Cost
Schedule
Performance
Risk
4
Government Insight Team in Plant

Figure 2. *GEMS impact on GPS Block IIF Program.*

among functional departments. Documents are now handled by cross-functional project teams, and the JPO business processes are well defined and easier to manage.

Facilitation of "process change." GEMS has allowed the creation of "virtual teams" that consist of both contractor and Block IIF IPT members working side by side in the contractor's plant and in multiple locations. Users quickly communicate issues throughout the group via the infrastructure. The organizational culture has become much more receptive to change, and information technology provides the necessary channels to disseminate information and facilitate change.

Stable configuration management. The heart of the GEMS system is the reference library, which holds most of the GPS program data. This data is cataloged by several factors (project, organization, type, subtype, date, etc.) for easy search and retrieval. The reference library is the single location for current copies of all program data; this eliminates having multiple versions of documents in circulation. Authorized individuals have fast access to the latest version of a document, including updates, from one location. The data is archived for safe, long-term storage.

Flexible implementation and usage. The nature of GPS JPO business forces GPS Block IIF team members to conduct business in many places other than their offices. The wide-area network and CITIS will permit users to view the same data from an equipped contractor's facility or remote JPO location. Based on their account privileges, these users have the same capabilities they have in their home office. Because of these capabilities, collocated GPS Block IIF team members in the contractor's plant are achieving unprecedented partnerships.

Management "insight" vs. oversight. The flexibility of GEMS permits GPS Block IIF IPT leads to task any GEMS user no matter where they are located. GEMS users will always have all the necessary tools and data to accomplish the work, even when they are not in the home office. The GPS Block IIF IPT leads have the same insight into job progress as if they were right down the hall.

Authenticating a GPS Block IIF Specification

Authenticating a specification is the process of reviewing the specification for accuracy and completeness by the government and contractor's engineering teams. An example of how GEMS is streamlining GPS operations is the authentication of the GPS Block IIF System Specification for the new GPS Block IIF satellite. This document serves as the technical backbone of the GPS Block IIF program and is the starting point for thousands of derived requirements.

Before GEMS, this process had always been long and costly. Paper copies of the specification were distributed and passed from one engineer to the next. The engineers had the continual task of coordinating comments, scheduling meetings, and checking status. Different groups of engineers would review issues that others had already resolved. Just the cost to reproduce the document would run into the thousands of dollars before a draft would be approved.

Because of the inability to track and manage the review process, the paper-based method of authenticating system specifications would normally take one to two years after contract award. Now, using GEMS, the reengineered process is significantly streamlined. Distribution to the entire GPS engineering team is virtually instantaneous. The reviewers can simultaneously see all comments to the document as soon as they are entered. Work-flows allow for management and tracking of the document throughout the review cycle. Key reviewers are notified if their input was overdue, thus helping the authentication review run smoothly.

Review managers no longer need to sit down with stacks of the same document with everyone's comments in the margins and try to consolidate them. Managers are now able to review, consolidate, approve, and transmit the results back to the contractor for incorporation.

The streamlined process using GEMS allowed the GPS Block IIF IPT to authenticate the IIF system specification in six months after contract award. The time savings not only saved substantial money but also has given both the government and contractors a solid baseline to build the GPS Block IIF program much sooner than would have been possible with the paper-based process. This in turn will help prevent requirements creep, which may save the government even more money in the future by preventing cost overruns.

Lessons Learned

- A key factor in the success of the GEMS project has been senior management commitment, a well-known success factor for any program that requires cultural change.
- User involvement in the early stages of the project helped ensure acceptance of the system.
- Variations in the desktop computer environment should be eliminated to the fullest extent possible. This will accelerate rollout and training and greatly reduce the burden on the system help desk.
- After the design and development of the new systems and processes are completed, management should resist the

desire to roll out the new systems too quickly for instant payback. A well-managed rollout to individual functional groups will allow for better and more targeted training and will contribute to a smoother implementation.

- Implementing electronic access to program data creates several issues related to the "ownership" of program data and who maintains the data of record. For the GPS Block IIF Program, this was resolved by the concept of a shared data environment between the contractor and the program office databases. Data in each database can be viewed by both government and contractor IPT members. Data to be retained by the program office can easily be transferred from the contractor's database to the GEMS reference library by IPT members over the electronic link.
- Credit should also be given to the implementation method. System development and deployment should not be implemented piecemeal during the process reengineering effort (risky integration), or a monolithic, all-at-once approach (too long to see results), but instead implemented in a modular, layered, bottom-up approach to minimize risk exposure and maximize flexibility.

Summary

The GEMS-based enterprise infrastructure fills the gap between ordinary office automation and the automation of JPO business processes. Using the DoD's JCALS infrastructure has allowed the IPTs of the NAVSTAR GPS JPO to concentrate on deploying modular process-based applications that can share enterprise data. Unlike systems that do not take advantage of continuous acquisition and lifecycle support (CALS) and industry standards, there are no constraints on data reuse, the longevity of data, or the amount or types of data (records, documents, or graphics) the system can manage, route, and warehouse. The organization retains its investment in applications, business processes, and data.

Because the GEMS business process applications that are developed on the DoD's JCALS infrastructure are modular and use CALS and industry standard data formats, the applications and process work-flows can be easily updated as the GPS JPO continuously improves its business processes. The applications can also be customized and deployed to other system program offices that use the JCALS infrastructure. GEMS has allowed the GPS Block IIF program and the GPS JPO to immediately implement acquisition reform initiatives by permitting fast, timely access to all unclassified program data.

Because of initiatives such as GEMS, the GPS Block IIF program won the 1995 Defense Standardization Program Award and the Secretary of the Air Force for Acquisition's Lightning Bolt Acquisition Reform Award for leading the way in acquisition reform excellence. The GPS JPO and the GPS Block IIF team continues to challenge themselves to do business better. ◆

About the Author

Lon Mehlman is a senior computer scientist with CSC in Moorestown, N.J.. He has over 15 years experience in the information technology industry. He has a bachelor's degree in economics/computer science and in sociology from the University of California at Los Angeles and a master's of business administration degree from Pepperdine University.

E-mail: mehlmald@gps1.laafb.af.mil.

Point of contact for GEMS:
Ernestine Reed
SMC/CZEC, 310.0363.2943
Los Angeles AFB, CA 90245

Web Addition

The following is excerpted from Emmett Paige's keynote address at the Tri-Ada conference in St. Louis in November 1997. The speech can be found in its entirety in CROSSTALK's Web Addition section at http://www.stsc.hill.af.mil/CrossTalk/crostalk.html.

The New Course for Ada in the DoD

Retired Lt. Gen. Emmett Paige Jr. (U.S. Army)
President, OAO Corporation

"There are numerous examples of superior technology failing to capture a consumer market. Military software has different requirements than consumer software. The DoD needs to learn what it can from the commercial sector and use those best practices that are applicable and appropriate for military requirements.

"The National Research Council study is right in asserting that Ada needs government support to survive. The Ada effort also needs help from industry to aggressively produce and market high-quality Ada tools and compilers. Perhaps most important is support from the education community. Without more and more well-educated scientists and engineers, not only will the Ada effort fail, but American technological superiority will become history."

STC ’98: If You Haven’t Registered, You Need to Now

Dana Dovenbarger
Software Technology Conference

Conference registration for the Tenth Annual Software Technology Conference (STC ’98) is well underway. If you haven’t already registered, register now. We anticipate over 3,500 participants will meet at the Salt Palace Convention Center in Salt Lake City, Utah during the week of April 19-23, 1998.

The U.S. Air Force, Army, Navy, Marine Corps, and the Defense Information Systems Agency (DISA) have again joined forces to co-sponsor STC ’98, the premier software technology conference in the Department of Defense (DoD). Once again, Utah State University Extension is the conference nonfederal co-sponsor.

The federal co-sponsors are Lt. Gen. David J. Kelley (DISA), Lt. Gen. William Campbell (U.S. Army), Dr. Helmut Hellwig (U.S. Air Force), Rear Adm. George Wagner (U.S. Navy), and Maj. Gen. Joseph Anderson (U.S. Marine Corps). The conference is hosted by Maj. Gen. Richard H. Roellig, commander of Hill Air Force Base, Utah and the Air Force Software Technology Support Center (STSC).

The theme for STC ’98 is “Knowledge Sharing – Global Information Networks.”

Hotel guestrooms are filling up fast. All reservations must be made through the Salt Lake City Visitor’s Bureau/STC Housing Bureau. Reservations for hotel quest rooms opened in August with the mailing of our Call for Speakers and Exhibitors brochure. Many early birds took advantage of being able to book their rooms then. We highly recommend that you send in your reservation form as soon as possible.

Conference registration and housing information and forms are available at http://www.stc98.org or by calling STC fax-on-demand 435-797-2358. You may also call the conference management at 801-777-7411 or DSN 777-7411.

The opening General Session on Monday afternoon will include keynote addresses given by Tony Valletta, acting assistant secretary of defense, and retired Vice Adm. Jerry Tuttle (U.S. Navy), president of ManTech Strategies Associates. Each of the co-sponsors also will address this session. Among the many other distinguished leaders in the DoD and industry executives who will speak at this year’s conference are Mark Schaeffer of the Office of the Under Secretary of Defense (Acquisition and Technology), Miriam Browning, director for information management, Office, Director for Information Systems Command, Control, Communications, and Computers, and Debra Filippi, deputy assistant chief of staff for command, control, communications, and computers systems integration.

Beginning in mid-January, as information was received from presentation speakers, presentation summaries and biographical information have been listed at our Web site, http://www.stc98.org. You will want to frequently visit this Web site prior to the conference to get the latest up-to-date information.

Conference management is coordinating military air flights to the conference from the San Diego and Washington, D.C. areas. If you are interested, it is essential that you contact conference management as soon as possible at 801-777-7411 or DSN 777-7411 or E-mail at wadel@software.hill.af.mil.

On Monday, April 20, 1998, conference participants may attend the “Birds-of-a-Feather” luncheon for which they have pre-registered. Participants should register for the luncheon they wish to attend by marking the appropriate box on the registration form. Designated luncheon discussion topics will include systems engineering, acquisition, and process improvement.

If we can be of further assistance, please call or E-mail. This is one conference that you do not want to miss. We will see you in April!

Dana Dovenbarger, Conference Manager
Lynne Wade, Assistant Conference Manager
Software Technology Support Center
OO-ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205
Voice: 801-777-7411 DSN 777-7411
Fax: 801-775-4932 DSN 775-4932
E-mail: dovenbar@oodiss1.hill.af.mil
wadel@software.hill.af.mil

CrossTALK

Sponsor	***Lt. Col. Joe Jarzombek*** 801-777-2435 DSN 777-2435 jarzombj@software.hill.af.mil
Publisher	***Reuel S. Alder*** 801-777-2550 DSN 777-2550 alderr@software.hill.af.mil
Managing Editor	***Tracy Stauder*** 801-777-9239 DSN 777-9239 staudert@software.hill.af.mil
Senior Editor	***Sandi Gaskin*** 801-777-9722 DSN 777-9722 gaskins@software.hill.af.mil
Graphics and Design	***Kent Hepworth*** 801-775-5555 ext. 3027 hepwortk@software.hill.af.mil
Associate Editor	***Lorin J. May*** 801-775-5555 ext. 3026 mayl@software.hill.af.mil
Editorial Assistant	***Bonnie May*** 801-775-5555 ext. 3023 mayb@software.hill.af.mil
Features Coordinator	***Heather Winward*** 801-775-5555 ext. 3028 winwardh@software.hill.af.mil
Customer Service	801-777-8045 DSN 777-8045 custserv@software.hill.af.mil
Fax	801-777-8069 DSN: 777-8069
STSC On-Line	http://www.stsc.hill.af.mil
CROSSTALK On-Line	http://www.stsc.hill.af.mil/Crosstalk/crostalk.html
ESIP On-Line	http://www.esip.hill.af.mil

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address:

Ogden ALC/TISE
7278 Fourth Street
Hill AFB, UT 84056-5205

E-mail:	custserv@software.hill.af.mil
Voice:	801-777-8045, DSN 777-8045
Fax:	801-777-8069, DSN 777-8069

Editorial Matters: Correspondence concerning *Letters to the Editor* or other editorial matters should be sent to the same address listed above to the attention of *CROSSTALK Editor* or send directly to the senior editor via the E-mail address also listed above.

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CROSSTALK editorial board prior to publication. Please follow the *Guidelines for CROSSTALK Authors*, available upon request. We do not pay for submissions. Articles published in CROSSTALK remain the property of the authors and may be submitted to other publications.

Reprints and Permissions: Most material in CROSSTALK may be reprinted at no charge. Coordinate reprint requests with CROSSTALK.

Trademarks and Endorsements: All product names referenced in this issue are trademarks of their companies. The mention of a product or business in CROSSTALK does not constitute an endorsement by the Software Technology Support Center (STSC), the Department of Defense, or any other government agency. The opinions expressed represent the viewpoints of the authors and are not necessarily those of the Department of Defense.

Coming Events: We often list conferences, seminars, symposiums, etc., that are of interest to our readers. There is no fee for this service, but we must receive the information at least 90 days before registration. Send an announcement to the CROSSTALK Editorial Department.

STSC On-Line Services: STSC On-Line Services can be reached on the Internet. World Wide Web access is at http://www.stsc.hill.af.mil. The STSC maintains a Gopher server at gopher://gopher.stsc.hill.af.mil/. Its ftp site may be reached at ftp://ftp.stsc.hill.af.mil. The Lynx browser or gopher server can also be reached using telnet at bbs.stsc.hill.af.mil or by modem at 801-774-6509 or DSN 775-3602. Call 801-777-7989 or DSN 777-7989 for assistance, or E-mail to portr@software.hill.af.mil.

Publications Available: The STSC provides various publications at no charge to the defense software community. Fill out the Request for STSC Services card in the center of this issue and mail or fax it to us. If the card is missing, call Customer Service at the numbers shown above, and we will send you a form or take your request by phone. The STSC sometimes has extra paper copies of back issues of CROSSTALK free of charge. If you would like a copy of the printed edition of this or another issue of CROSSTALK, or would like to subscribe, please contact the customer service address listed above.

The ***Software Technology Support Center*** was established at Ogden Air Logistics Center (AFMC) by Headquarters U.S. Air Force to help Air Force software organizations identify, evaluate, and adopt technologies that will improve the quality of their software products, their efficiency in producing them, and their ability to accurately predict the cost and schedule of their delivery. CROSSTALK is assembled, printed, and distributed by the Defense Automated Printing Service, Hill AFB, UT 84056. CROSSTALK is distributed without charge to individuals actively involved in the defense software development process.

Barbershop Training: No Results for Less Money

I've been a professional corporate trainer for nearly seven years, although I didn't realize it until recently. Technically, my title has been "singer in a semiprofessional quartet," but it turns out we've been doing exactly what trainers do all along. I'm not talking about training in programming languages or technologies—we are best qualified to train people in your organization's next program or edict. But before our sales pitch, I'll use my own experiences to explain the importance of training.

I have no experience with software training, but I received great training when I sold home electronics for a national retailer. I realize that retail sales and software engineering are worlds apart—in retail sales, the effectiveness of front-line workers has a huge impact on profitability—but there are some important similarities.

For example, you may think selling TVs and stereos involves nothing more than bluffing about product features and implying that most products tend to explode into flames if you don't buy an extended service plan. However, there are many technical aspects, such as how to steal sales from co-workers and how to avoid lawsuits while insinuating to customers that the competition, for entertainment, likes to push old ladies in front of moving buses. So for this *part-time* job, the company gave me four weeks of full-time paid training in every imaginable sales technique and product feature, knowing it would pay big dividends. It matters little that I quit before they could make back their training investment, because we shouldn't carry this analogy too far—after all, the retail industry has a big problem with employee turnover.

Yet, despite great returns such as these, many managers are tight with their training dollars, and sometimes ignore the basics. For example, I was fresh out of college when I got a job creating artsy how-to books. I realize the publishing industry is also world's apart from software programming—publishing is a deadline-, coordination-, and process-intensive field where any oversight can become an expensive embarrassment—but if you use your imagination, there's a lesson here. The following near-verbatim excerpt shows my former boss' mistaken attitude about training, although I'm certain software managers would never be this shortsighted:

Boss: "We need you to quickly create these five books that cover topics you know nothing about. Use these programs you've never seen and follow our strict processes, which we won't explain."
Me: "… ah, all right. Who can show me how to—"
Boss: "—Oh, so we get to hold your hand, 'Mr. Qualified'? Let's see some initiative! Well, that does it for your training—if you have any questions, feel free to rudely interrupt something important."

The problem? Many managers believe that if employees have skill and "initiative" (manager code for "clairvoyance"), they don't need training. But at my next publishing job, I learned that even seasoned employees need regular training. Among other duties, I had to get "old-school" people to use new software and computers. At first I thought they would resist learning new ways to do their jobs, but after a cycle of training, mentoring, and hands-on experience, within a year they were all proficient on the new system, which they subsequently ignored. And then we were all laid off.

So I can't overemphasize the long-term benefits of training—which brings me to my sales pitch. For years, businesses and professional groups have hired my quartet to perform at banquets and parties. See if what we've been doing resembles any session where your organization has covered a company program, process, or policy.

Picture a roomful of professionals who are there only because their bosses made them go. Someone in a high position stands up and drones on for a few minutes, occasionally blurting out words like "vision" or "excellence." Eventually, the performers (read: "trainers") are introduced, who then put on a well-choreographed presentation, interspersed with corny jokes. The person who paid for this sits in rapt attention, while the rest nervously glance at their watches and try not bang their heads on the table too loudly when they fall asleep.

Sound familiar? To complete the transition from "singers" to "trainers," we'll just need to start handing out binders, which no one ever reads anyway. But our binders won't be stuffed away and forgotten—they'll be made of chocolate. Our half-hour of "training" will provide the measurable results of many of the training sessions you've been through: the trainers get paid, your boss is satisfied, and everybody leaves the room and continues with business as usual. Except that we're quick, cheap, and sometimes we're even asked to do an encore! Hire us now, and we'll throw in a free recording of our barbershop rendition of "Stairway to Heaven." —Lorin May

Got an idea for BACKTALK? Send an E-mail to mayl@software.hill.af.mil